2

IDA MEMORANDUM REPORT M-361

EXAMPLE LEVEL 1 Ada/SQL SYSTEM SOFTWARE

Bill Brykczynski
Fred Friedman
Kerry Hilliard
Audrey Hook

September 1987

Prepared for
Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This Memorandum Report is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A196 632

1a REPORT SECURITY CLASSIFICATION Unclassified	**1b RESTRICTIVE MARKINGS**
2a SECURITY CLASSIFICATION AUTHORITY	**3 DISTRIBUTION/AVAILABILITY OF REPORT** Public release/distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Memorandum Report M-361	**5 MONITORING ORGANIZATION REPORT NUMBER(S)**

6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses	**6b OFFICE SYMBOL** IDA	**7a NAME OF MONITORING ORGANIZATION** OUSDA, DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311		**7b ADDRESS (City, State, and Zip Code)** 1801 N. Beauregard St. Alexandria, VA 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION Defense Logistics Agency	**8b OFFICE SYMBOL (if applicable)** DLA-Z	**9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER** MDA 903 84 C 0031

8c ADDRESS (City, State, and Zip Code)
Cameron Station
Alexandria, VA 22304-6100

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
		T-T5-423	

11 TITLE (Include Security Classification)
Example Level 1 Ada/SQL System Software (U)

12 PERSONAL AUTHOR(S)
Bill Brykczynski, Fred Friedman, Kerry Hilliard, Audrey A. Hook

13a TYPE OF REPORT	**13b TIME COVERED**	**14 DATE OF REPORT (Year, Month, Day)**	**15 PAGE COUNT**
Final	FROM ______ TO ______	1987 September	144

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Ada programming language; structured query language (SQL); binding specifications; relational database systems; software engineering; database tables; interface specifications; administrative processing systems; UNIFY.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

IDA Memorandum Report M-361 contains the source code for the demonstration software which implements the specification found in IDA Memorandum Report M-360, *Level 1 Ada/SQL Database Language Interface User's Guide*. M-361 will be used to provide the actual means of accessing the relational database UNIFY with the Ada language. This software demonstrates an interface between Ada and the database language SQL as implemented by the UNIFY database package. An interface between Ada application programs and the UNIFY database was identified as a necessary capability which must be demonstrated to show that Ada applications can be integrated with the DLA operational environment. This interface implements a subset of the ANSI SQL standard which reflects the current implementation of SQL by the UNIFY. However, this software can be easily upgraded to support an ANSI implementation of SQL when it is acquired by the DLA.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	**21 ABSTRACT SECURITY CLASSIFICATION** Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Audrey A. Hook	**22b TELEPHONE (Include area code)** (703) 824-5501	**22c OFFICE SYMBOL** IDA/CSED

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

IDA MEMORANDUM REPORT M-361

EXAMPLE LEVEL 1 Ada/SQL SYSTEM SOFTWARE

Bill Brykczynski
Fred Friedman
Kerry Hilliard
Audrey Hook

September 1987

IDA
INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-T5-423

CONTENTS

PREFACE

The purpose of IDA Memorandum Report M-361, *Example Level 1 Ada/SQl System Software*, is to forward data developed in the course of an investigation. This Memorandum Report presents the actual software source code which implements the specification found in IDA Memorandum Report M-361, *Level 1 Ada/SQL Database Language Interface User's Guide*.

This document partially fulfills the objective of Task Order T-T5-423, Defense Logistics Agency Information Systems, which is to provide a capability for accessing a relational database from the Ada language. IDA Memorandum Report M-361 will be used to provide the actual means of accessing the relational database UNIFY with the Ada language. As a Memorandum Report, M-361 is directed to those users of Ada/SQL within the Defense Logistics Agency.

1. Introduction

This report documents the demonstration software provided to the Defense Logistics Agency, in partial fulfillment of IDA task T-T5-423 "DLA Information Systems." This software demonstrates an interface between the Ada programming language and the database language SQL as implemented by the UNIFY database package. An interface between Ada application programs and the UNIFY database was identified as a necessary capability that must be demonstrated to show that Ada applications can be integrated with the DLA operational environment. This interface implements a sub-set of the ANSI SQL standard that reflects the current implementation of SQL by the UNIFY database package. However, this software can easily be upgraded to support an ANSI implementation of SQL when it is acquired by DLA.

2. Package UNIFY_DEFINITIONS

```
with TEXT_IO;
use  TEXT_IO;

package UNIFY_DEFINITIONS is

  type TYPE_EXEC is (UNKNOWN, DELETE, FETCH, INSERT, SELEC, UPDATE);
  type STATUS_FILE is (NOT_CREATED, CREATED, OPEN, DONE, CLOSED, DELETED);
  type TYPE_RESULT is (NONE, SUCCESS, NOT_FOUND, NOT_UNIQUE, ERROR);

  type CURSOR_NAME_RECORD is
    record
      IN_FILE     : FILE_TYPE;
      IN_STAT     : STATUS_FILE := NOT_CREATED;
      OUT_FILE    : FILE_TYPE;
      OUT_STAT    : STATUS_FILE := NOT_CREATED;
      ERR_FILE    : FILE_TYPE;
      ERR_STAT    : STATUS_FILE := NOT_CREATED;
      SEQ_NUM     : NATURAL := 0;
      SEQ_STR     : STRING (1..10) := (others => ' ');
      SEQ_LEN     : NATURAL := 0;
      EXEC_TYPE   : TYPE_EXEC := UNKNOWN;
      RESULT_TYPE : TYPE_RESULT := NONE;
      GOT_DATA    : BOOLEAN := FALSE;
      BUFFER      : STRING (1..1024) := (others => ' ');
      BUF_LEN     : NATURAL := 0;
      BUF_PTR     : NATURAL := 0;
      BUF_ROW     : NATURAL := 0;
    end record;

  type CURSOR_NAME is access CURSOR_NAME_RECORD;

  NOT_FOUND_ERROR : exception;
  UNIFY_ERROR     : exception;
  UNIQUE_ERROR    : exception;

end UNIFY_DEFINITIONS;
```

3. Package UNIFY_VARIABLES

```
with UNIFY_DEFINITIONS;
use  UNIFY_DEFINITIONS;

package UNIFY_VARIABLES is

  SEQ_NUMBER            : NATURAL := 0;
  GOT_PID               : BOOLEAN := FALSE;
  INPUT_FILE_NAME_LEN   : constant NATURAL := 16;
  INPUT_FILE_NAME       : STRING (1..INPUT_FILE_NAME_LEN) := "ADA_SQL_IN_00000";
  OUTPUT_FILE_NAME_LEN  : constant NATURAL := 17;
  OUTPUT_FILE_NAME      : STRING (1..OUTPUT_FILE_NAME_LEN) :=
                          "ADA_SQL_OUT_00000";
  ERROR_FILE_NAME_LEN   : constant NATURAL := 17;
  ERROR_FILE_NAME       : STRING (1..ERROR_FILE_NAME_LEN) :=
                          "ADA_SQL_ERR_00000";
  A_NEW_LINE_1          : constant CHARACTER := ascii.cr;
  A_NEW_LINE            : constant STRING := ascii.cr & ascii.lf;
  A_NEW_LINE_LEN        : constant NATURAL := 2;
  FETCH_CURSOR          : CURSOR_NAME := null;
  COLUMN                : STRING (1..1024) := (others => ' ');
  COLUMN_LEN            : NATURAL := 0;

end UNIFY_VARIABLES;
```

4. Package UNIFY_ERRORS

```
package UNIFY_ERRORS is

  ERROR_BUFFER       : STRING (1..500) := (others => ' ');
  ERROR_BUFFER_LEN   : NATURAL := 0;

end UNIFY_ERRORS;
```

5. Package Specification UNIFY_SUBROUTINES

```
with TEXT_IO, UNIFY_DEFINITIONS, UNIFY_VARIABLES, UNIFY_ERRORS;
use  TEXT_IO, UNIFY_DEFINITIONS, UNIFY_VARIABLES, UNIFY_ERRORS;

package UNIFY_SUBROUTINES is

  procedure SEQ_NUM_TO_STRING
           (SEQ_NUM     : in NATURAL;
            STR         : in out STRING;
            LEN         : in out NATURAL);

  procedure READ_FOR_ERRORS
           (CURSOR : in CURSOR_NAME;
            ERR    : out NATURAL);

  procedure READ_A_LINE
           (TYPE_FILE  : in FILE_TYPE;
            STAT_FILE  : in out STATUS_FILE;
            BUF        : in out STRING;
            BUF_LEN    : out NATURAL);

  procedure SET_UP_OUT_FILE
           (CURSOR : in CURSOR_NAME;
            ERR    : out NATURAL);

  procedure RESPONSE
           (TYP    : in TYPE_EXEC;
            ERR    : out NATURAL);

  procedure IS_NUMERIC
           (BUF          : in STRING;
            PTR0         : in out NATURAL;
            PTR9         : in NATURAL;
            TRUE_FALSE   : out BOOLEAN;
            ZERO         : out BOOLEAN);

  procedure IS_STRING
           (BUF          : in STRING;
            PTR0         : in out NATURAL;
            PTR9         : in NATURAL;
            CMPR         : in STRING;
            TRUE_FALSE   : out BOOLEAN);

  function NEXT_COLUMN
           (CURSOR   : in CURSOR_NAME)
            return BOOLEAN;

  procedure ADD_PIDNO
```

```
           (STR : in out STRING;
            PID : in INTEGER);

end UNIFY_SUBROUTINES;
```

6. Package Body UNIFY_SUBROUTINES

```
package body UNIFY_SUBROUTINES is

-------------------------------------------------------------------------
--
-- SEQ_NUM_TO_STRING

  procedure SEQ_NUM_TO_STRING
           (SEQ_NUM      : in NATURAL;
            STR          : in out STRING;
            LEN          : in out NATURAL) is

    TSTR   : STRING (1..10) := (others => ' ');
    TLEN   : NATURAL := 0;

  begin
    TLEN := NATURAL'IMAGE (SEQ_NUM)'LENGTH;
    TSTR (1..TLEN) := NATURAL'IMAGE (SEQ_NUM);
    LEN := 0;
    for I in 1..TLEN loop
      if TSTR (I) in '0'..'9' then
        LEN := LEN + 1;
        STR (LEN) := TSTR (I);
      end if;
    end loop;
  end SEQ_NUM_TO_STRING;

-------------------------------------------------------------------------
--
-- READ_FOR_ERRORS
--
-- open and read the error file created by the execution of the unify query.
-- ERR = 0 if nothing in the file
-- ERR = 1 if "There were no records selected."
-- ERR = 2 if any other error
-- if any errors stuff the whole message in the error message buffer

  procedure READ_FOR_ERRORS
           (CURSOR : in CURSOR_NAME;
            ERR    : out NATURAL) is
    LINES_READ : NATURAL := 0;
    PTR        : NATURAL := 0;
    LEN        : NATURAL := 0;

  begin
    ERR := 0;
    OPEN (CURSOR.ERR_FILE, IN_FILE,
            ERROR_FILE_NAME (1..ERROR_FILE_NAME_LEN) & "." &
            CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN));
    CURSOR.ERR_STAT := OPEN;
    ERROR_BUFFER_LEN := 0;
    loop
      PTR := ERROR_BUFFER_LEN + 1;
      LEN := 0;
      READ_A_LINE (CURSOR.ERR_FILE, CURSOR.ERR_STAT,
                   ERROR_BUFFER (PTR .. ERROR_BUFFER'LAST), LEN);
      if LEN > 0 then
        LEN := LEN - ERROR_BUFFER_LEN;
      end if;
      if CURSOR.ERR_STAT = DONE then
        DELETE (CURSOR.ERR_FILE);
        --CLOSE (CURSOR.ERR_FILE);
        CURSOR.ERR_STAT := DELETED;
        return;
      end if;
      ERROR_BUFFER_LEN := ERROR_BUFFER_LEN + LEN;
      if LEN > 0 then
        LINES_READ := LINES_READ + 1;
        if LINES_READ = 1 and then ERROR_BUFFER (1..ERROR_BUFFER_LEN) =
                    "There were no records selected." then
          ERR := 1;
        else
          ERR := 2;
        end if;
        ERROR_BUFFER (ERROR_BUFFER_LEN + 1 .. ERROR_BUFFER_LEN + A_NEW_LINE_LEN)
                     := A_NEW_LINE;
        ERROR_BUFFER_LEN := ERROR_BUFFER_LEN + A_NEW_LINE_LEN;
      end if;
    end loop;
  end READ_FOR_ERRORS;

----------------------------------------------------------------------------
--
-- READ_A_LINE
--
-- given a cursor name, return the next line in the buffer

  procedure READ_A_LINE
           (TYPE_FILE  : in FILE_TYPE;
            STAT_FILE  : in out STATUS_FILE;
            BUF        : in out STRING;
            BUF_LEN    : out NATURAL) is

  begin
    BUF_LEN := 0;
    if STAT_FILE = OPEN then
      GET_LINE (TYPE_FILE, BUF, BUF_LEN);
    end if;

  exception
    when END_ERROR => STAT_FILE := DONE;
                      BUF_LEN := 0;
  end READ_A_LINE;

-----------------------------------------------------------------------
--
-- SET_UP_OUT_FILE
--
-- open and read the output file created by the execution of the unify query.
-- ERR = 0 if the file consists of a title line then a ---- line
-- ERR = 1 if file is empty - not_found_error
-- ERR = 2 if any other error - unify_error
-- ERR = 3 mulfiple rows on select - unique_error
-- if any errors stuff the whole message in the error message buffer

  procedure SET_UP_OUT_FILE
           (CURSOR : in CURSOR_NAME;
            ERR    : out NATURAL) is

    LINES_READ  : NATURAL := 0;
    PTR         : NATURAL := 0;
    LEN         : NATURAL := 0;
    SELECT_THIS : BOOLEAN := FALSE;
    D_BUF       : STRING (1..1024) := (others => ' ');
    D_LEN       : NATURAL := 0;

  begin
    ERR := 0;
    OPEN (CURSOR.OUT_FILE, IN_FILE,
            OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN) & "." &
            CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN));
    CURSOR.OUT_STAT := OPEN;
    CURSOR.BUF_LEN := 0;
    CURSOR.BUF_PTR := 0;
    CURSOR.BUF_ROW := 0;
    ERROR_BUFFER_LEN := 0;
    loop
      if LINES_READ >= 2 and CURSOR.EXEC_TYPE = SELEC then
        SELECT_THIS := TRUE;
      else
  SELECT_THIS := FALSE;
end if;
exit when CURSOR.EXEC_TYPE = FETCH and LINES_READ = 2;
PTR := ERROR_BUFFER_LEN + 1;
LEN := 0;
if SELECT_THIS then
  if LINES_READ = 2 then
    READ_A_LINE (CURSOR.OUT_FILE, CURSOR.OUT_STAT,
                 CURSOR.BUFFER, CURSOR.BUF_LEN);
    LEN := 0;
    if CURSOR.BUF_LEN > 0 then
      CURSOR.BUF_ROW := CURSOR.BUF_ROW + 1;
      LEN := CURSOR.BUF_LEN;
      CURSOR.BUF_PTR := 1;
    end if;
  else
    READ_A_LINE (CURSOR.OUT_FILE, CURSOR.OUT_STAT,
                 D_BUF, D_LEN);
    LEN := 0;
    if D_LEN > 0 then
      CURSOR.BUF_ROW := CURSOR.BUF_ROW + 1;
      LEN := D_LEN;
    end if;
  end if;
else
  READ_A_LINE (CURSOR.OUT_FILE, CURSOR.OUT_STAT,
               ERROR_BUFFER (PTR .. ERROR_BUFFER'LAST), LEN);
  if LEN > 0 then
    LEN := LEN - ERROR_BUFFER_LEN;
  end if;
end if;
if CURSOR.OUT_STAT = DONE then
  DELETE (CURSOR.OUT_FILE);
  --CLOSE (CURSOR.OUT_FILE);
  CURSOR.OUT_STAT := DELETED;
  case CURSOR.EXEC_TYPE is
    when UNKNOWN => ERR := 2;
    when DELETE  => RESPONSE (DELETE, ERR);
    when FETCH   => ERR := 2;
    when INSERT  => RESPONSE (INSERT, ERR);
    when SELEC   => if CURSOR.BUF_ROW < 1 then
                      ERR := 2;
                    elsif CURSOR.BUF_ROW > 1 then
                      ERR := 3;
                    end if;
    when UPDATE  => RESPONSE (UPDATE, ERR);
  end case;
  return;
end if;
if LEN > 0 then
```

```
        LINES_READ := LINES_READ + 1;
        if not SELECT_THIS then
          ERROR_BUFFER_LEN := ERROR_BUFFER_LEN + LEN;
          ERROR_BUFFER (ERROR_BUFFER_LEN + 1 .. ERROR_BUFFER_LEN +
                        A_NEW_LINE_LEN) := A_NEW_LINE;
          ERROR_BUFFER_LEN := ERROR_BUFFER_LEN + A_NEW_LINE_LEN;
        end if;
      end if;
    end loop;
  end SET_UP_OUT_FILE;

-----------------------------------------------------------------------
--
-- RESPONSE
--
-- given an execute type and the response from unify in the error buffer
-- see if it's a valid delete, update or insert response
--  err = 0 valid
--  err = 1 not found error
--  err = 2 unify error

  procedure RESPONSE
           (TYP    : in TYPE_EXEC;
            ERR    : out NATURAL) is

    TRUE_FALSE : BOOLEAN := FALSE;
    ZERO       : BOOLEAN := FALSE;
    PTR        : NATURAL := 1;

  begin
    ERR := 0;
    IS_NUMERIC (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN, TRUE_FALSE, ZERO);
    if not TRUE_FALSE then
      ERR := 2;
      return;
    end if;
    if ZERO then
      ERR := 1;
      return;
    end if;
    IS_STRING (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN, "record(s)", TRUE_FALSE);
    if not TRUE_FALSE then
      ERR := 2;
      return;
    end if;
    case TYP is
      when DELETE => IS_STRING (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN,
                     "selected,", TRUE_FALSE);
      when INSERT => IS_STRING (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN,
                     "added.", TRUE_FALSE);
      when UPDATE => IS_STRING (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN,
                     "updated.", TRUE_FALSE);
      when others => null;
    end case;
    if not TRUE_FALSE then
      ERR := 2;
      return;
    end if;
    if TYP /= DELETE then
      return;
    end if;
    IS_NUMERIC (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN, TRUE_FALSE, ZERO);
    if not TRUE_FALSE then
      ERR := 2;
      return;
    end if;
    if ZERO then
      ERR := 1;
      return;
    end if;
    IS_STRING (ERROR_BUFFER, PTR, ERROR_BUFFER_LEN, "record(s) deleted",
               TRUE_FALSE);
    if not TRUE_FALSE then
      ERR := 2;
      return;
    end if;
  end RESPONSE;

------------------------------------------------------------------------
--
-- IS_NUMERIC
--
-- given a buffer and a pointer to the current spot and the end
-- look for a number first.  Ignore leading spaces or new_lines
-- then check for 0-9 until encountering a space or new_line
-- if only 0s are encountered it's a zero

  procedure IS_NUMERIC
           (BUF          : in STRING;
            PTR0         : in out NATURAL;
            PTR9         : in NATURAL;
            TRUE_FALSE   : out BOOLEAN;
            ZERO         : out BOOLEAN) is

    LEADING_SPACE : BOOLEAN := TRUE;
    C             : CHARACTER := ' ';

  begin
    TRUE_FALSE := TRUE;
    ZERO := TRUE;
    loop
      if PTR0 > PTR9 then
        return;
      end if;
      C := BUF (PTR0);
      case C is
        when ' '          =>  if not LEADING_SPACE then
                                return;
                              end if;
        when A_NEW_LINE_1 =>  if not LEADING_SPACE then
                                return;
                              end if;
                              PTR0 := PTR0 + A_NEW_LINE_LEN;
        when '0'          =>  LEADING_SPACE := FALSE;
        when '1'..'9'     =>  ZERO := FALSE;
                              LEADING_SPACE := FALSE;
        when others       =>  TRUE_FALSE := FALSE;
                              ZERO := FALSE;
                              return;
      end case;
      PTR0 := PTR0 + 1;
    end loop;
  end IS_NUMERIC;

-----------------------------------------------------------------------
--
-- IS_STRING
--
-- given a buffer and a pointer to the current spot and the end
-- look for a string that matches "cmpr".  Ignore spaces and new_lines
-- in the buffer and the compare string.

  procedure IS_STRING
           (BUF          : in STRING;
            PTR0         : in out NATURAL;
            PTR9         : in NATURAL;
            CMPR         : in STRING;
            TRUE_FALSE   : out BOOLEAN) is

    B             : CHARACTER := ' ';
    C             : CHARACTER := ' ';
    CP            : NATURAL   := 1;

  begin
    TRUE_FALSE := FALSE;
    loop
      exit when PTR0 > PTR9 or CP > CMPR'LAST;
      loop
        exit when PTR0 > PTR9;
        B := BUF (PTR0);
```

```
          if B = ' ' then
            PTR0 := PTR0 + 1;
          elsif B = A_NEW_LINE (1) then
            PTR0 := PTR0 + A_NEW_LINE_LEN - 1;
          else
            exit;
          end if;
        end loop;
        loop
          exit when CP > CMPR'LAST;
          C := CMPR (CP);
          if C = ' ' then
            CP := CP + 1;
          elsif C = A_NEW_LINE (1) then
            CP := CP + A_NEW_LINE_LEN - 1;
          else
            exit;
          end if;
        end loop;
        B := BUF (PTR0);
        C := CMPR (CP);
        if B in 'a'..'z' then
          B := CHARACTER'VAL (CHARACTER'POS (B) - 32);
        end if;
        if C in 'a'..'z' then
          C := CHARACTER'VAL (CHARACTER'POS (C) - 32);
        end if;
        if B /= C then
          return;
        end if;
        PTR0 := PTR0 + 1;
        CP := CP + 1;
      end loop;
      if CP > CMPR'LAST then
        TRUE_FALSE := TRUE;
      end if;
    end IS_STRING;

----------------------------------------------------------------------------
--
-- NEXT_COLUMN

    function NEXT_COLUMN
             (CURSOR   : in CURSOR_NAME)
              return BOOLEAN is

      LEADING_SPACES : BOOLEAN := TRUE;
      BUF_PTR_START  : NATURAL := CURSOR.BUF_PTR;
      BUF_PTR_END    : NATURAL := CURSOR.BUF_PTR;
```

```
  begin
    COLUMN_LEN := 0;
    if (CURSOR.EXEC_TYPE = FETCH and CURSOR.OUT_STAT /= OPEN) or
        CURSOR.RESULT_TYPE /= SUCCESS or
        CURSOR.BUF_PTR > CURSOR.BUF_LEN or
        CURSOR.BUF_LEN < 1 then
      return FALSE;
    end if;
    loop
      exit when CURSOR.BUFFER (CURSOR.BUF_PTR) = '|';
      exit when CURSOR.BUF_PTR > CURSOR.BUF_LEN;
      if not LEADING_SPACES or else CURSOR.BUFFER (CURSOR.BUF_PTR) /= ' ' then
        LEADING_SPACES := FALSE;
        COLUMN_LEN := COLUMN_LEN + 1;
        COLUMN (COLUMN_LEN) := CURSOR.BUFFER (CURSOR.BUF_PTR);
      end if;
      CURSOR.BUF_PTR := CURSOR.BUF_PTR + 1;
    end loop;
    BUF_PTR_END := CURSOR.BUF_PTR;
    if CURSOR.BUF_PTR <= CURSOR.BUF_LEN and then
       CURSOR.BUFFER (CURSOR.BUF_PTR) = '|' then
      CURSOR.BUF_PTR := CURSOR.BUF_PTR + 1;
    end if;
    if BUF_PTR_END <= BUF_PTR_START then
      return FALSE;
    end if;
    loop
      exit when COLUMN_LEN < 1;
      exit when COLUMN (COLUMN_LEN) /= ' ';
      COLUMN_LEN := COLUMN_LEN - 1;
    end loop;
    if COLUMN_LEN < 1 then
      COLUMN_LEN := 1;
      COLUMN (COLUMN_LEN) := ' ';
    end if;
    return TRUE;
  end NEXT_COLUMN;

------------------------------------------------------------------------
--
-- ADD_PIDNO

  procedure ADD_PIDNO
           (STR : in out STRING;
            PID : in INTEGER) is

    TSTR : STRING (1..10) := (others => ' ');
    TEND : INTEGER := 0;
    TBEG : INTEGER := 0;
    PSTR : STRING (1..5) := (others => '0');
    II   : INTEGER := 0;

  begin
    for I in STR'FIRST..STR'LAST loop
      II := I;
      exit when STR (I) = ascii.nul;
    end loop;
    TEND := INTEGER'IMAGE (PID)'LAST;
    TSTR (1..TEND) := INTEGER'IMAGE (PID);
    for I in 1..TEND loop
      TBEG := I;
      exit when TSTR(I) in '0'..'9';
    end loop;
    PSTR ( (5 - (TEND - TBEG))..5) := TSTR (TBEG..TEND);
    if II + 4 <= STR'LAST then
      STR (II..II+4) := PSTR (1..5);
    end if;
  end ADD_PIDNO;

end UNIFY_SUBROUTINES;
```

7. Package UNIFY_ROUTINES

```
with TEXT_IO, SYSTEM, UNIFY_DEFINITIONS, UNIFY_VARIABLES, UNIFY_SUBROUTINES,
     UNIFY_ERRORS;
use  TEXT_IO, SYSTEM, UNIFY_DEFINITIONS, UNIFY_VARIABLES, UNIFY_SUBROUTINES,
     UNIFY_ERRORS;

package UNIFY_ROUTINES is

  procedure CREATE_ADA_SQL_INPUT_FILE
           (CURSOR : in out CURSOR_NAME);

  procedure EXECUTE_ADA_SQL_FILE
           (CURSOR : in out CURSOR_NAME);

  procedure FETCH
           (CURSOR : in CURSOR_NAME);

  generic
    type USER_TYPE is (<>);
  procedure INTEGER_AND_ENUMERATION_INTO
           (VAR : out USER_TYPE);

  generic
    type USER_TYPE is digits <>;
  procedure FLOAT_INTO
```

```
          (VAR : out USER_TYPE);

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT
                 (C : CHARACTER)
                  return COMPONENT_TYPE is <>;
  prc ɜdure UNCONSTRAINED_STRING_INTO
          (VAR  : out USER_TYPE;
           LAST : out INDEX_TYPE);

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT
                 (C : CHARACTER)
                  return COMPONENT_TYPE is <>;
  procedure CONSTRAINED_STRING_INTO
          (VAR  : out USER_TYPE;
           LAST : out INDEX_TYPE);

end UNIFY_ROUTINES;

package body UNIFY_ROUTINES is

---------------------------------------------------------------------------
--
-- CREATE_ADA_SQL_INPUT_FILE
--
-- this routine creates and opens a new file "ada_sql_in_pid.seq" where pid
-- is the process id for this program and seq is a sequential number assigned
-- to the input files.  This file is where the UNIFY query will be written
-- out to for later execution.

  procedure CREATE_ADA_SQL_INPUT_FILE
          (CURSOR : in out CURSOR_NAME) is

    MY_PID : INTEGER := 0;
    function CGETPID return INTEGER;
    pragma Interface (C, CGETPID);

  begin
    if not GOT_PID then
      INPUT_FILE_NAME  (INPUT_FILE_NAME_LEN - 4)  := ascii.nul;
      OUTPUT_FILE_NAME (OUTPUT_FILE_NAME_LEN - 4) := ascii.nul;
      ERROR_FILE_NAME  (ERROR_FILE_NAME_LEN - 4)  := ascii.nul;
      MY_PID := CGETPID;
      ADD_PIDNO (INPUT_FILE_NAME, MY_PID);
      ADD_PIDNO (OUTPUT_FILE_NAME, MY_PID);
      ADD_PIDNO (ERROR_FILE_NAME, MY_PID);
      GOT_PID := TRUE;
    end if;
    SEQ_NUMBER := SEQ_NUMBER + 1;
    CURSOR.SEQ_NUM := SEQ_NUMBER;
    SEQ_NUM_TO_STRING (SEQ_NUMBER, CURSOR.SEQ_STR, CURSOR.SEQ_LEN);
    CREATE (CURSOR.IN_FILE, OUT_FILE,
            INPUT_FILE_NAME (1..INPUT_FILE_NAME_LEN) & "." &
            CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN));
    CURSOR.IN_STAT := OPEN;
    PUT_LINE (CURSOR.IN_FILE, "lines 99999999999");
  end CREATE_ADA_SQL_INPUT_FILE;

------------------------------------------------------------------------
--
-- EXECUTE_ADA_SQL_FILE
--
-- close the ada sql input file that was created in create_ada_sql_input_file,
-- call unix to execute the file and put the output in a file called
-- "ada_sql_out_pid.seq" and the error output in a file called
-- "ada_sql_err_pid.seq" where pid is the process id for this program and
-- seq is a sequential number assigned to the output files.
-- Delete the file ada_sql_in_pid.seq
-- Open the error file and the output file, if we have an error other than
-- "There were no records selected." raise the UNIFY_ERROR exception.
-- If "There were no records selected." and this is an execute for a fetch
-- just wait and raise NOT_FOUND_ERROR exception when the user makes his
-- first call to FETCH.  If it's any other kind of execute raise
-- NOT_FOUND_ERROR exception now.  If it's a delete make sure the comment
-- is "x record(s) selected, x record(s) deleted.".  If it's an insert make
-- sure the comment is "x record(s) added.".  If it's an update make sure
-- the comment is "x record(s) updated.".

  procedure EXECUTE_ADA_SQL_FILE
           (CURSOR : in out CURSOR_NAME) is

    subtype ADDRESS is SYSTEM.ADDRESS;
    procedure CSYSTEM (STR : ADDRESS);
    pragma Interface (C, CSYSTEM);
    TMP     : STRING (1..200) := (others => ' ');
    TMP_LEN : NATURAL := 0;
    ERR     : NATURAL := 0;

  begin
    FETCH_CURSOR := CURSOR;
    CLOSE (CURSOR.IN_FILE);
    CURSOR.IN_STAT := CLOSED;
    TMP_LEN := 4 + INPUT_FILE_NAME_LEN + 1 + CURSOR.SEQ_LEN + 2 +
               OUTPUT_FILE_NAME_LEN + 1 + CURSOR.SEQ_LEN + 3 +
               ERROR_FILE_NAME_LEN + 1 + CURSOR.SEQ_LEN + 1;
    TMP (1..TMP_LEN) := "SQL " & INPUT_FILE_NAME (1..INPUT_FILE_NAME_LEN) &
                        "." & CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN) & " >" &
                        OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN) & "." &
                        CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN) &
                        " 2>" & ERROR_FILE_NAME (1..ERROR_FILE_NAME_LEN) &
                        "." & CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN) & ascii.nul;
    --PUT_LINE ("going to unify:");
    --PUT_LINE ("      " & TMP (1..TMP_LEN));
    CSYSTEM (TMP'ADDRESS);
    CURSOR.OUT_STAT := CREATED;
    CURSOR.ERR_STAT := CREATED;
    OPEN (CURSOR.IN_FILE, IN_FILE,
            INPUT_FILE_NAME (1..INPUT_FILE_NAME_LEN) & "." &
            CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN));
    DELETE (CURSOR.IN_FILE);
    --CLOSE (CURSOR.IN_FILE);
    CURSOR.IN_STAT := DELETED;
    READ_FOR_ERRORS (CURSOR, ERR);
    if ERR = 0 then
      SET_UP_OUT_FILE (CURSOR, ERR);
    else
      OPEN (CURSOR.OUT_FILE, IN_FILE,
            OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN) & "." &
            CURSOR.SEQ_STR (1..CURSOR.SEQ_LEN));
      CURSOR.OUT_STAT := OPEN;
      DELETE (CURSOR.OUT_FILE);
      --CLOSE (CURSOR.OUT_FILE);
      CURSOR.OUT_STAT := DELETED;
    end if;
    case ERR is
      when 0 => CURSOR.RESULT_TYPE := SUCCESS;
      when 1 => CURSOR.RESULT_TYPE := NOT_FOUND;
      when 2 => CURSOR.RESULT_TYPE := ERROR;
      when 3 => CURSOR.RESULT_TYPE := NOT_UNIQUE;
      when others => null;
    end case;
    if ERR = 1 and CURSOR.EXEC_TYPE /= FETCH then
      raise NOT_FOUND_ERROR;
    elsif ERR = 2 then
      raise UNIFY_ERROR;
    elsif ERR = 3 then
      raise UNIQUE_ERROR;
    end if;
  end EXECUTE_ADA_SQL_FILE;

-------------------------------------------------------------------------
--
-- FETCH
```

```
--
-- user is ready to do a fetch, save the cursor to do the following intos
-- from and read the next line of the output file.  If none or if the
-- query had been unsuccessful then raise NOT_FOUND_ERROR exception.
-- when reaching the end of the file, delete it.

  procedure FETCH
           (CURSOR : in CURSOR_NAME) is
  begin
    FETCH_CURSOR := CURSOR;
    READ_A_LINE (FETCH_CURSOR.OUT_FILE, FETCH_CURSOR.OUT_STAT,
                 FETCH_CURSOR.BUFFER, FETCH_CURSOR.BUF_LEN);
    if FETCH_CURSOR.OUT_STAT /= OPEN or else
               FETCH_CURSOR.RESULT_TYPE /= SUCCESS then
      if IS_OPEN (FETCH_CURSOR.OUT_FILE) then
        DELETE (FETCH_CURSOR.OUT_FILE);
        --CLOSE (FETCH_CURSOR.OUT_FILE);
        FETCH_CURSOR.OUT_STAT := DELETED;
      end if;
      raise NOT_FOUND_ERROR;
    end if;
    FETCH_CURSOR.BUF_ROW := FETCH_CURSOR.BUF_ROW + 1;
    FETCH_CURSOR.BUF_PTR := 1;
  end FETCH;

---------------------------------------------------------------------------
--
-- INTEGER_AND_ENUMERATION_INTO

  procedure INTEGER_AND_ENUMERATION_INTO
           (VAR : out USER_TYPE) is

    TMP : INTEGER := 0;
    TVAR : USER_TYPE;

  begin
    if not NEXT_COLUMN (FETCH_CURSOR) then
      raise NOT_FOUND_ERROR;
    end if;
    TMP := INTEGER'VALUE (COLUMN (1..COLUMN_LEN));
    TVAR := USER_TYPE'VAL (TMP);
    VAR := USER_TYPE'VAL (TMP);
  end INTEGER_AND_ENUMERATION_INTO;

---------------------------------------------------------------------------
--
-- FLOAT_INTO

  procedure FLOAT_INTO
           (VAR : out USER_TYPE) is
    package GET_FLOAT is new FLOAT_IO (USER_TYPE);
    LAST : POSITIVE := 1;

  begin
    if not NEXT_COLUMN (FETCH_CURSOR) then
      raise NOT_FOUND_ERROR;
    end if;
    GET_FLOAT.GET (COLUMN (1..COLUMN_LEN), VAR, LAST);
    if LAST /= COLUMN_LEN then
      raise DATA_ERROR;
    end if;
  end FLOAT_INTO;

---------------------------------------------------------------------------
--
-- UNCONSTRAINED_STRING_INTO

  procedure UNCONSTRAINED_STRING_INTO
           (VAR  : out USER_TYPE;
            LAST : out INDEX_TYPE) is

    V   : INDEX_TYPE := VAR'FIRST;

  begin
    if not NEXT_COLUMN (FETCH_CURSOR) then
      raise NOT_FOUND_ERROR;
    end if;
    LAST := VAR'FIRST + INDEX_TYPE (COLUMN_LEN - 1);
    for I in 1..COLUMN_LEN loop
      VAR (V) := CONVERT_CHARACTER_TO_COMPONENT (COLUMN (I));
      if V < INDEX_TYPE'LAST then
        V := V + 1;
      end if;
    end loop;
  end UNCONSTRAINED_STRING_INTO;

---------------------------------------------------------------------------
--
-- CONSTRAINED_STRING_INTO

  procedure CONSTRAINED_STRING_INTO
           (VAR  : out USER_TYPE;
            LAST : out INDEX_TYPE) is

    V   : INDEX_TYPE := VAR'FIRST;

  begin
    if not NEXT_COLUMN (FETCH_CURSOR) then
      raise NOT_FOUND_ERROR;
    end if;
    LAST := VAR'FIRST + INDEX_TYPE (COLUMN_LEN - 1);
    for I in 1..COLUMN_LEN loop
      VAR (V) := CONVERT_CHARACTER_TO_COMPONENT (COLUMN (I));
      if V < INDEX_TYPE'LAST then
        V := V + 1;
      end if;
    end loop;
  end CONSTRAINED_STRING_INTO;

end UNIFY_ROUTINES;
```

8. Package TEXT_PRINT

```
with TEXT_IO;
  use TEXT_IO;

package TEXT_PRINT is

  type LINE_TYPE is limited private;

  type BREAK_TYPE is (BREAK, NO_BREAK);

  type PHANTOM_TYPE is private;

  procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE);

  procedure SET_LINE(LINE : in LINE_TYPE);

  function CURRENT_LINE return LINE_TYPE;

  procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL);
  procedure SET_INDENT(INDENT : in NATURAL);

  procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
                                    INDENT : in INTEGER);
  procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER);

  function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE;

  procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                         START_PHANTOM,
                         END_PHANTOM   : in PHANTOM_TYPE);

  procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE);

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
```

```
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure PRINT_LINE(FILE : in FILE_TYPE);
procedure PRINT_LINE(LINE : in LINE_TYPE);
procedure PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure BLANK_LINE(FILE : in FILE_TYPE);
procedure BLANK_LINE(LINE : in LINE_TYPE);
procedure BLANK_LINE;

generic
  type NUM is range <>;
package INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);

  procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

end INTEGER_PRINT;

generic
  type NUM is digits <>;
package FLOAT_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
```

```
    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);

    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

  end FLOAT_PRINT;

  NULL_PHANTOM : constant PHANTOM_TYPE;

  LAYOUT_ERROR : exception renames TEXT_IO.LAYOUT_ERROR;

private

  type PHANTOM_TYPE is access STRING;

  type LINE_REC(LENGTH : INTEGER) is
    record
      USED_YET            : BOOLEAN := FALSE;
      INDENT              : INTEGER := 0;
      CONTINUATION_INDENT : INTEGER := 2;
      BREAK               : INTEGER := 1;
      INDEX               : INTEGER := 1;
      DATA                : STRING(1..LENGTH);
      START_PHANTOM,
      END_PHANTOM         : PHANTOM_TYPE := NULL_PHANTOM;
    end record;

  type LINE_TYPE is access LINE_REC;

  NULL_PHANTOM : constant PHANTOM_TYPE := new STRING'("");

end TEXT_PRINT;

package body TEXT_PRINT is

  DEFAULT_LINE : LINE_TYPE;

  procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE) is
  begin
    LINE := new LINE_REC(LENGTH);
  end CREATE_LINE;

  procedure SET_LINE(LINE : in LINE_TYPE) is
  begin
  DEFAULT_LINE := LINE;
end SET_LINE;

function CURRENT_LINE return LINE_TYPE is
begin
  return DEFAULT_LINE;
end CURRENT_LINE;

procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL) is
begin
  if INDENT >= LINE.LENGTH then
    raise LAYOUT_ERROR;
  end if;
  if LINE.INDEX = LINE.INDENT + 1 then
    for I in 1..INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
    LINE.INDEX := INDENT + 1;
  end if;
  LINE.INDENT := INDENT;
end SET_INDENT;

procedure SET_INDENT(INDENT : in NATURAL) is
begin
  SET_INDENT(DEFAULT_LINE,INDENT);
end SET_INDENT;

procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
                                  INDENT : in INTEGER) is
begin
  if LINE.INDENT + INDENT >= LINE.LENGTH or else LINE.INDENT + INDENT < 0
      then
    raise LAYOUT_ERROR;
  end if;
  LINE.CONTINUATION_INDENT := INDENT;
end SET_CONTINUATION_INDENT;

procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER) is
begin
  SET_CONTINUATION_INDENT(DEFAULT_LINE,INDENT);
end SET_CONTINUATION_INDENT;

function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE is
begin
  return new STRING'(S);
end MAKE_PHANTOM;

procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                       START_PHANTOM,
                       END_PHANTOM   : in PHANTOM_TYPE) is
```

```
begin
  LINE.START_PHANTOM := START_PHANTOM;
  LINE.END_PHANTOM := END_PHANTOM;
end SET_PHANTOMS;

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE) is
begin
  SET_PHANTOMS(DEFAULT_LINE,START_PHANTOM,END_PHANTOM);
end SET_PHANTOMS;

procedure PRINT(FILE : in FILE_TYPE;
                LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
  NEW_BREAK, NEW_INDEX : INTEGER;
begin
  if LINE.INDEX + ITEM'LENGTH + LINE.END_PHANTOM'LENGTH > LINE.LENGTH + 1
      then
    if LINE.INDENT + LINE.CONTINUATION_INDENT + LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK + ITEM'LENGTH > LINE.LENGTH then
      raise LAYOUT_ERROR;
    end if;
    if ITEM = " " and then LINE.END_PHANTOM.all = "" then
      return;
    end if;
    PUT_LINE(FILE,LINE.DATA(1..LINE.BREAK-1) & LINE.END_PHANTOM.all);
    for I in 1..LINE.INDENT + LINE.CONTINUATION_INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
    NEW_BREAK := LINE.INDENT + LINE.CONTINUATION_INDENT + 1;
    NEW_INDEX := NEW_BREAK + LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK;
    LINE.DATA(NEW_BREAK..NEW_INDEX) := LINE.START_PHANTOM.all &
        LINE.DATA(LINE.BREAK..LINE.INDEX);
    LINE.BREAK := NEW_BREAK;
    LINE.INDEX := NEW_INDEX;
  end if;
  NEW_INDEX := LINE.INDEX + ITEM'LENGTH;
  LINE.DATA(LINE.INDEX..NEW_INDEX-1) := ITEM;
  LINE.INDEX := NEW_INDEX;
  if BRK = BREAK then
    LINE.BREAK := NEW_INDEX;
  end if;
  LINE.USED_YET := TRUE;
end PRINT;

procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
```

```
  PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in STRING; BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if LINE.INDEX /= LINE.INDENT + 1 then
    PUT_LINE(FILE,LINE.DATA(1..LINE.INDEX-1));
  end if;
  for I in 1..LINE.INDENT loop
    LINE.DATA(I) := ' ';
  end loop;
  LINE.INDEX := LINE.INDENT + 1;
  LINE.BREAK := LINE.INDEX;
end PRINT_LINE;

procedure PRINT_LINE(FILE : in FILE_TYPE) is
begin
  PRINT_LINE(FILE,DEFAULT_LINE);
end PRINT_LINE;

procedure PRINT_LINE(LINE : in LINE_TYPE) is
begin
  PRINT_LINE(CURRENT_OUTPUT,LINE);
end PRINT_LINE;

procedure PRINT_LINE is
begin
  PRINT_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
end PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if LINE.USED_YET then
    NEW_LINE(FILE);
  end if;
end BLANK_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE) is
```

```
begin
  BLANK_LINE(FILE,DEFAULT_LINE);
end BLANK_LINE;

procedure BLANK_LINE(LINE : in LINE_TYPE) is
begin
  BLANK_LINE(CURRENT_OUTPUT,LINE);
end BLANK_LINE;

procedure BLANK_LINE is
begin
  BLANK_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
end BLANK_LINE;

package body INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
    S : STRING(1..NUM'WIDTH);
    L : NATURAL;
  begin
    PRINT(S,L,ITEM);
    PRINT(FILE,LINE,S(1..L),BRK);
  end PRINT;

  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
    S : constant STRING := NUM'IMAGE(ITEM);
    F : NATURAL := S'FIRST; -- Bug in DG Compiler -- S'FIRST /= 1 ! ! ! ! ! !
```

```
      L : NATURAL;
    begin
      if S(F) = ' ' then
        F := F + 1;
      end if;
      if TO'LENGTH < S'LAST - F + 1 then
        raise LAYOUT_ERROR;
      end if;
      L := TO'FIRST + S'LAST - F;
      TO(TO'FIRST..L) := S(F..S'LAST);
      LAST := L;
    end PRINT;

  end INTEGER_PRINT;

  package body FLOAT_PRINT is

    package NUM_IO is new FLOAT_IO(NUM);
      use NUM_IO;

    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
      S : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
      L : NATURAL;
    begin
      PRINT(S,L,ITEM);
      PRINT(FILE,LINE,S(1..L),BRK);
    end PRINT;

    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
    end PRINT;

    procedure PRINT(LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
    end PRINT;

    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
    end PRINT;
```

```
procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
  S         : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
  EXP       : INTEGER;
  E_INDEX   : NATURAL := S'LAST - DEFAULT_EXP;
  DOT_INDEX : NATURAL := DEFAULT_FORE + 1;
  L         : NATURAL;
begin
  PUT(S,ITEM);
  EXP := INTEGER'VALUE(S(E_INDEX+1..S'LAST));
  if EXP >= 0 then
    if EXP <= DEFAULT_AFT-1 then
      S(DOT_INDEX..DOT_INDEX+EXP-1) := S(DOT_INDEX+1..DOT_INDEX+EXP);
      S(DOT_INDEX+EXP) := '.';
      for I in E_INDEX..S'LAST loop
        S(I) := ' ';
      end loop;
    end if;
  else -- EXP < 0
    if EXP >= - ( DEFAULT_EXP + 1 ) then
      S(DEFAULT_EXP+2..S'LAST) := S(1..S'LAST-DEFAULT_EXP-1);
      for I in 1..DEFAULT_EXP+1 loop
        S(I) := ' ';
      end loop;
      E_INDEX := S'LAST + 1;
      DOT_INDEX := DOT_INDEX + DEFAULT_EXP + 1;
      L := DOT_INDEX+EXP;
      for I in reverse L+1..DOT_INDEX loop
        case S(I-1) is
          when ' '    => S(I) := '0';
          when '-'    => S(I-2) := '-'; S(I) := '0';
          when others => S(I) := S(I-1);
        end case;
      end loop;
      S(L) := '.';
      case S(L-1) is
        when ' '    => S(L-1) := '0';
        when '-'    => S(L-2) := '-'; S(L-1) := '0';
        when others => null;
      end case;
    end if;
  end if;
  for I in reverse 1..E_INDEX-1 loop
    exit when S(I) /= '0' or else S(I-1) = '.';
    S(I) := ' ';
  end loop;
  L := 0;
  for I in S'RANGE loop
    if S(I) /= ' ' then
      L := L + 1;
      TO(L) := S(I);
```

```
        end if;
      end loop;
      LAST := L;
    exception
      when CONSTRAINT_ERROR =>
        raise LAYOUT_ERROR;
    end PRINT;

  end FLOAT_PRINT;

end TEXT_PRINT;
```

9. Package ADA_SQL_FUNCTION

```
with SYSTEM;
package DATABASE is -- ***** FOR NOW, FOR TESTING PURPOSES
  type INTG             is range SYSTEM.MIN_INT .. SYSTEM.MAX_INT;
  type DOUBLE_PRECISION is digits SYSTEM.MAX_DIGITS;
end DATABASE;

with DATABASE;
package ADA_SQL_FUNCTIONS is

  INTERNAL_ERROR : exception;

  type SQL_OPERATION is
  ( O_AVG                , O_MAX         , O_MIN       , O_SUM               ,
    O_UNARY_PLUS         , O_UNARY_MINUS , O_PLUS      , O_MINUS             ,
    O_TIMES              , O_DIVIDE      , O_EQ        , O_NE                ,
    O_LT                 , O_GT          , O_LE        , O_GE                ,
    O_BETWEEN            , O_AND         , O_IS_IN     , O_OR                ,
    O_NOT                , O_LIKE        , O_AMPERSAND , O_SELEC             ,
    O_SELECT_DISTINCT    , O_ASC         , O_DESC      , O_TABLE_COLUMN_LIST ,
    O_COUNT_STAR         , O_NULL_OP     , O_STAR      , O_NOT_IN            ,
    O_VALUES             , O_DECLAR );

  type SQL_OBJECT       is private;
  type TYPED_SQL_OBJECT is private;
  type TABLE_NAME       is private;
  type TABLE_LIST       is private;
  type INSERT_ITEM      is private;
  type CURSOR_NAME      is private;
  type DATABASE_NAME    is private;

  NULL_SQL_OBJECT : constant SQL_OBJECT;

  procedure INITIATE_TEST; -- ***** ONLY FOR TESTING
```

```
-- constant literal value generator

  generic
    type RESULT_TYPE is private;
    VALUE : in RESULT_TYPE;
  function CONSTANT_LITERAL return RESULT_TYPE;

-- conversion routines for SQL objects

  function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT;

  function R_CONVERT ( R : TYPED_SQL_OBJECT ) return SQL_OBJECT
   renames L_CONVERT;

  function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT;

  package CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT;

    function R_CONVERT ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT;

    function R_CONVERT ( R : TABLE_NAME ) return SQL_OBJECT renames L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME;

    function L_CONVERT ( L : TABLE_LIST ) return SQL_OBJECT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_LIST;

    function L_CONVERT ( L : INSERT_ITEM ) return SQL_OBJECT;

    function CONVERT_R ( R : SQL_OBJECT ) return INSERT_ITEM;

  end CONVERT;

-- conversion routines for user types

  -- ***** instantiate these as L_CONVERT, then rename as R_CONVERT

  generic
    type USER_TYPE is (<>);
  function INTEGER_AND_ENUMERATION_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
```

```
    type USER_TYPE is digits <>;
  function FLOAT_CONVERT ( VAR : USER_TYPE ) return SQL_OBJECT;

  generic
    type INDEX_TYPE is range <>;
    type USER_TYPE is array ( INDEX_TYPE range <> ) of CHARACTER;
  function UNCONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
    type INDEX_TYPE is range <>;
    type USER_TYPE is array ( INDEX_TYPE ) of CHARACTER;
  function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_COMPONENT_TO_CHARACTER ( C : COMPONENT_TYPE )
     return CHARACTER is <>;
  function UNCONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  -- ***** must generate CONVERT_COMPONENT_TO_CHARACTER

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_COMPONENT_TO_CHARACTER ( C : COMPONENT_TYPE )
     return CHARACTER is <>;
  function CONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

-- column and table name routines

  generic
    GIVEN_NAME : in STANDARD.STRING;
  package NAME_PACKAGE is

    generic
      type SQL_OBJECT_TYPE is private;
      with function CONVERT_R ( R : SQL_OBJECT ) return SQL_OBJECT_TYPE is <>;
    function COLUMN_OR_TABLE_NAME return SQL_OBJECT_TYPE;

    generic
    function TABLE_NAME_WITH_COLUMN_LIST ( COLUMNS : SQL_OBJECT )
     return TABLE_NAME;
  end NAME_PACKAGE;

  -- ***** must generate routines for table.column (define record structure)

  -- ***** must generate package for correlation.column and correlation.table

-- value specification routines

  generic
    type USER_TYPE is private;
    type RESULT_TYPE is private;
    with function L_CONVERT ( L : USER_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return RESULT_TYPE is <>;
  function INDICATOR_FUNCTION ( VAL : USER_TYPE ) return RESULT_TYPE;

-- generic operation routines

  generic
    GIVEN_OPERATION : in SQL_OPERATION;
    type L_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : L_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function UNARY_OPERATION ( L : L_TYPE ) return TYPE_R;

  generic
    GIVEN_OPERATION : in SQL_OPERATION;
    type L_TYPE is private;
    type R_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : L_TYPE ) return SQL_OBJECT is <>;
    with function R_CONVERT ( R : R_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function BINARY_OPERATION ( L : L_TYPE ; R : R_TYPE ) return TYPE_R;

-- set function routines

  -- ***** must also generate STAR_TYPE is '*'; function COUNT ( STAR_TYPE )
  -- ***** instantiate COUNT_STAR for DATABASE.INTG or untyped

  generic
    type TYPE_R is private;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function COUNT_STAR return TYPE_R;

  -- instantiate UNARY_OPERATION for O_AVG, O_MAX, O_MIN, O_SUM

-- value expression routines

  -- instantiate UNARY_OPERATION for O_UNARY_PLUS, O_UNARY_MINUS
  -- instantiate BINARY_OPERATION for O_PLUS, O_MINUS, O_TIMES, O_DIVIDE

  -- ***** generate CONVERT_TO package for type conversions, calling CONVERT_R
  --  to set correct result type

-- comparison predicate routines

  -- instantiate BINARY_OPERATION for O_EQ, O_NE, O_LT, O_GT, O_LE, O_GE

-- between predicate routines

  -- instantiate BINARY_OPERATION for O_BETWEEN

  -- instantiate BINARY_OPERATION for O_AND

-- in predicate routines

  -- instantiate BINARY_OPERATION for O_IS_IN

    -- special case if <in value list> has one element

  -- instantiate BINARY_OPERATION for O_OR

    -- different instantiations for first and following ORs

  -- instantiate UNARY_OPERATION for O_NOT

-- like predicate routines

  -- instantiate BINARY_OPERATION for O_LIKE

  -- instantiate UNARY_OPERATION for O_NOT

-- search condition routines

  -- instantiate BINARY_OPERATION for O_AND, O_OR

  -- instantiate UNARY_OPERATION for O_NOT

-- from clause routines

  -- instantiate BINARY_OPERATION for O_AMPERSAND

-- group by clause routines

  -- instantiate BINARY_OPERATION for O_AMPERSAND

-- subquery routines

  generic
```

```
    SELECT_TYPE : in SQL_OPERATION;
    type WHAT_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : WHAT_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function SELECT_LIST_SUBQUERY
           ( WHAT     : WHAT_TYPE;
             FROM     : TABLE_LIST;
             WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
             GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
             HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R;

  generic
    SELECT_TYPE : in SQL_OPERATION;
    type TYPE_R is private;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function STAR_SUBQUERY
           ( FROM     : TABLE_LIST;
             WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
             GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
             HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R;

-- query specification routines

  -- instantiate appropriate subquery routines

  -- also instantiate BINARY_OPERATION for O_AMPERSAND

-- close routine

  procedure CLOSE ( CURSOR : in out CURSOR_NAME );

-- declare cursor routines

  procedure DECLAR
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     SQL_OBJECT := NULL_SQL_OBJECT );

  -- instantiate BINARY_OPERATION for O_AMPERSAND

  -- instantiate UNARY_OPERATION for O_ASC and O_DESC

-- delete routines

  procedure DELETE_FROM
            ( TABLE : in TABLE_NAME;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT );

-- fetch and into routines
```

```
  procedure FETCH ( CURSOR : in out CURSOR_NAME );

  generic
    type USER_TYPE is (<>);
  procedure INTEGER_AND_ENUMERATION_INTO ( VAR : out USER_TYPE );

  generic
    type USER_TYPE is digits <>;
  procedure FLOAT_INTO ( VAR : out USER_TYPE );

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
     return COMPONENT_TYPE is <>;
  procedure UNCONSTRAINED_STRING_INTO
            ( VAR : out USER_TYPE ; LAST : out INDEX_TYPE );

  -- ***** must generate CONVERT_CHARACTER_TO_COMPONENT

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
     return COMPONENT_TYPE is <>;
  procedure CONSTRAINED_STRING_INTO
            ( VAR : out USER_TYPE ; LAST : out INDEX_TYPE );

-- insert into routines

  procedure INSERT_INTO
            ( TABLE : in TABLE_NAME;
              WHAT  : in INSERT_ITEM );

  -- instantiate BINARY_OPERATION for O_AMPERSAND

  -- see table name routines for table ( column list )

  function VALUES return INSERT_ITEM;

  -- instantiate BINARY_OPERATION for O_LE and O_AND

-- open routine

  procedure OPEN ( CURSOR : in out CURSOR_NAME );

-- select statement routines
```

```
  -- see above for fetch and into routines

  generic
    SELECT_TYPE : in SQL_OPERATION;
    type WHAT_TYPE is private;
    with function L_CONVERT ( L : WHAT_TYPE ) return SQL_OBJECT is <>;
  procedure SELECT_LIST_SELECT
            ( WHAT     : in WHAT_TYPE;
              FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT );

  generic
    SELECT_TYPE : in SQL_OPERATION;
  procedure STAR_SELECT
            ( FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT );

-- update routines

  procedure UPDATE
            ( TABLE : in TABLE_NAME;
              SET   : in SQL_OBJECT;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT );

  -- instantiate BINARY_OPERATION for O_AND

  -- instantiate BINARY_OPERATION for O_LE

private

  type DATABASE_NAME is access STANDARD.STRING;
  type ACCESS_STRING is access STANDARD.STRING;

  type SQL_VALUE_KIND is ( INTEGER , FLOAT , STRING );

  type SQL_VALUE ( KIND : SQL_VALUE_KIND := INTEGER ) is
    record
      case KIND is
        when INTEGER =>
          INTEGER : DATABASE.INTG;
        when FLOAT =>
          FLOAT : DATABASE.DOUBLE_PRECISION;
        when STRING =>
          STRING : ACCESS_STRING;
      end case;
    end record;
  type SQL_OBJECT_KIND is ( NAME , VALUE , OPERATION );

  type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND );
  type TYPED_SQL_OBJECT is access SQL_OBJECT_RECORD;
  type SQL_OBJECT  is new TYPED_SQL_OBJECT;
  type TABLE_NAME  is new TYPED_SQL_OBJECT;
  type TABLE_LIST  is new TYPED_SQL_OBJECT;
  type INSERT_ITEM is new TYPED_SQL_OBJECT;

  type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND ) is
    record
      ACROSS : SQL_OBJECT;
      case KIND is
        when NAME =>
          NAME : DATABASE_NAME;
        when VALUE =>
          VALUE : SQL_VALUE;
        when OPERATION =>
          OPERATION : SQL_OPERATION;
          OPERANDS  : SQL_OBJECT;
      end case;
    end record;

  NULL_SQL_OBJECT : constant SQL_OBJECT := null;

  type CURSOR_NAME is new SQL_OBJECT; -- ***** FOR NOW, FOR TESTING PURPOSES

end ADA_SQL_FUNCTIONS;

with ADA_SQL_FUNCTIONS;
package CURSOR_DEFINITION is
  subtype CURSOR_NAME is ADA_SQL_FUNCTIONS.CURSOR_NAME;
end CURSOR_DEFINITION;

with TEXT_PRINT;
 use TEXT_PRINT;
package body ADA_SQL_FUNCTIONS is

  INDENT : STANDARD.INTEGER;

  package DOUBLE_PRECISION_PRINT is new
   FLOAT_PRINT ( DATABASE.DOUBLE_PRECISION );

  package INTG_PRINT is new INTEGER_PRINT ( DATABASE.INTG );

  use DOUBLE_PRECISION_PRINT , INTG_PRINT;

  LINE : LINE_TYPE;

-- declarations for print routines (since some are recursive and mutually
--  recursive)

  procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT );
  procedure SHOW_ALL_SET_FUNCTION    ( S : in SQL_OBJECT );
  procedure SHOW_VALUE_EXPRESSION    ( S : in SQL_OBJECT );
  procedure SHOW_BETWEEN_PREDICATE   ( S : in SQL_OBJECT );
  procedure SHOW_IN_VALUE_LIST       ( S : in SQL_OBJECT );
  procedure SHOW_LIKE_PREDICATE      ( S : in SQL_OBJECT );
  procedure SHOW_SEARCH_CONDITION    ( S : in SQL_OBJECT );
  procedure SHOW_TABLE_EXPRESSION    ( S : in SQL_OBJECT );
  procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT );
  procedure SHOW_SELECT_LIST         ( S : in SQL_OBJECT );
  procedure SHOW_ORDER_BY_CLAUSE     ( S : in SQL_OBJECT );
  procedure SHOW_INSERT_VALUE_LIST   ( S : in SQL_OBJECT );
  procedure SHOW_SET_CLAUSES         ( S : in SQL_OBJECT );
  procedure SHOW_COMPARISON_PREDICATE
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING );
  procedure SHOW_IN_PREDICATE
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING );

  procedure INITIATE_TEST is -- ***** FOR TESTING ONLY
  begin
    CREATE_LINE ( LINE , 79 );
    SET_LINE ( LINE );
    SET_CONTINUATION_INDENT ( 7 );
  end INITIATE_TEST;

-- constant literal value generator

  function CONSTANT_LITERAL return RESULT_TYPE is
  begin
    return VALUE;
  end CONSTANT_LITERAL;

-- conversion routines for SQL objects

  function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT is
  begin
    return SQL_OBJECT ( L );
  end L_CONVERT;

  function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT is
  begin
    return TYPED_SQL_OBJECT ( R );
  end CONVERT_R;

  package body CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT is
    begin
```

```
      return L;
    end L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME is
    begin
      return TABLE_NAME ( R );
    end CONVERT_R;

    function L_CONVERT ( L : TABLE_LIST ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_LIST is
    begin
      return TABLE_LIST ( R );
    end CONVERT_R;

    function L_CONVERT ( L : INSERT_ITEM ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return INSERT_ITEM is
    begin
      return INSERT_ITEM ( R );
    end CONVERT_R;

  end CONVERT;

-- conversion routines for user types

  function INTEGER_AND_ENUMERATION_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
  begin
    return
     new SQL_OBJECT_RECORD'
         ( VALUE , null , ( INTEGER , USER_TYPE'POS ( VAR ) ) );
  end INTEGER_AND_ENUMERATION_CONVERT;

  function FLOAT_CONVERT ( VAR : USER_TYPE ) return SQL_OBJECT is
  begin
    return
     new SQL_OBJECT_RECORD'
         ( VALUE , null , ( FLOAT , DATABASE.DOUBLE_PRECISION ( VAR ) ) );
  end FLOAT_CONVERT;

  function UNCONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
    S : ACCESS_STRING := new STANDARD.STRING ( 1 .. VAR'LENGTH );
  begin
    S.all := STANDARD.STRING ( VAR );
    return new SQL_OBJECT_RECORD' ( VALUE , null , ( STRING , S ) );
  end UNCONSTRAINED_CHARACTER_STRING_CONVERT;

  function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
    S : ACCESS_STRING := new STANDARD.STRING ( 1 .. VAR'LENGTH );
  begin
    S.all := STANDARD.STRING ( VAR );
    return new SQL_OBJECT_RECORD' ( VALUE , null , ( STRING , S ) );
  end CONSTRAINED_CHARACTER_STRING_CONVERT;

  function UNCONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
    S : ACCESS_STRING := new STANDARD.STRING ( 1.. VAR'LENGTH );
    I : POSITIVE      := 1;
  begin
    for J in VAR'RANGE loop
      S(I) := CONVERT_COMPONENT_TO_CHARACTER ( VAR(J) );
      I := I + 1;
    end loop;
    return new SQL_OBJECT_RECORD' ( VALUE , null , ( STRING , S ) );
  end UNCONSTRAINED_STRING_CONVERT;

  function CONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
    S : ACCESS_STRING := new STANDARD.STRING ( 1.. VAR'LENGTH );
    I : POSITIVE      := 1;
  begin
    for J in VAR'RANGE loop
      S(I) := CONVERT_COMPONENT_TO_CHARACTER ( VAR(J) );
      I := I + 1;
    end loop;
    return new SQL_OBJECT_RECORD' ( VALUE , null , ( STRING , S ) );
  end CONSTRAINED_STRING_CONVERT;

-- column and table name routines

  package body NAME_PACKAGE is

    NAME_P : constant DATABASE_NAME := new STANDARD.STRING' ( GIVEN_NAME );

    function COLUMN_OR_TABLE_NAME return SQL_OBJECT_TYPE is
    begin
```

```
      return CONVERT_R ( new SQL_OBJECT_RECORD' ( NAME , null , NAME_P ) );
    end COLUMN_OR_TABLE_NAME;

    function TABLE_NAME_WITH_COLUMN_LIST ( COLUMNS : SQL_OBJECT )
     return TABLE_NAME is
      N : SQL_OBJECT := new SQL_OBJECT_RECORD' ( NAME , COLUMNS , NAME_P );
    begin
      return
       new SQL_OBJECT_RECORD' ( OPERATION , null , O_TABLE_COLUMN_LIST , N );
    end TABLE_NAME_WITH_COLUMN_LIST;

  end NAME_PACKAGE;

-- value specification routines

  function INDICATOR_FUNCTION ( VAL : USER_TYPE ) return RESULT_TYPE is
  begin
    return CONVERT_R ( L_CONVERT ( VAL ) );
  end INDICATOR_FUNCTION;

-- generic operation routines

  function UNARY_OPERATION ( L : L_TYPE ) return TYPE_R is
  begin
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD'
       ( OPERATION , null , GIVEN_OPERATION , L_CONVERT ( L ) ) );
  end UNARY_OPERATION;

  function BINARY_OPERATION ( L : L_TYPE ; R : R_TYPE ) return TYPE_R is
    LEFT : SQL_OBJECT := L_CONVERT ( L );
  begin
    LEFT.ACROSS := R_CONVERT ( R );
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD' ( OPERATION , null , GIVEN_OPERATION , LEFT ) );
  end BINARY_OPERATION;

-- set function routines

  function COUNT_STAR return TYPE_R is
  begin
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD' ( OPERATION , null , O_COUNT_STAR , null ) );
  end COUNT_STAR;

-- subquery routines
```

```
function NEW_TAIL ( L , R : SQL_OBJECT ) return SQL_OBJECT is
begin
  if R = null then
    L.ACROSS :=
     new SQL_OBJECT_RECORD' ( OPERATION , null , O_NULL_OP , null );
  else
    L.ACROSS := R;
  end if;
  return L.ACROSS;
end NEW_TAIL;

function BUILD_SELECT
         ( SELECT_TYPE                : SQL_OPERATION;
           WHAT                       : SQL_OBJECT;
           FROM                       : TABLE_LIST;
           WHERE , GROUP_BY , HAVING  : SQL_OBJECT )
 return SQL_OBJECT is
  TAIL : SQL_OBJECT :=
   NEW_TAIL
   ( NEW_TAIL
     ( NEW_TAIL ( SQL_OBJECT ( FROM ) , WHERE ) , GROUP_BY ) , HAVING );
begin
  WHAT.ACROSS := SQL_OBJECT ( FROM );
  return new SQL_OBJECT_RECORD' ( OPERATION , null , SELECT_TYPE , WHAT );
end BUILD_SELECT;

function SELECT_LIST_SUBQUERY
         ( WHAT     : WHAT_TYPE;
           FROM     : TABLE_LIST;
           WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
           GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
           HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R is
begin
  return
   CONVERT_R
   ( BUILD_SELECT
     ( SELECT_TYPE,
       L_CONVERT ( WHAT ) , FROM , WHERE , GROUP_BY , HAVING ) );
end SELECT_LIST_SUBQUERY;

function STAR_SUBQUERY
         ( FROM     : TABLE_LIST;
           WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
           GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
           HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R is
begin
  return
   CONVERT_R
   ( BUILD_SELECT
     ( SELECT_TYPE,
         new SQL_OBJECT_RECORD' ( OPERATION , null , O_STAR , null ),
         FROM , WHERE , GROUP_BY , HAVING ) );
  end STAR_SUBQUERY;

-- print routines

  -- 5.6.1 <value specification>

  procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT ) is
  begin
    case S.VALUE.KIND is
      when INTEGER => PRINT ( S.VALUE.INTEGER );
      when FLOAT   => PRINT ( S.VALUE.FLOAT );
      when STRING  => PRINT ( "'" & S.VALUE.STRING.all & "'" );
    end case;
  end SHOW_VALUE_SPECIFICATION;

  -- 5.8.3 <all set function>

  procedure SHOW_ALL_SET_FUNCTION ( S : in SQL_OBJECT ) is
  begin
    case S.OPERATION is
      when O_AVG  => PRINT ( "AVG( " );
      when O_MAX  => PRINT ( "MAX( " );
      when O_MIN  => PRINT ( "MIN( " );
      when O_SUM  => PRINT ( "SUM( " );
      when others => raise INTERNAL_ERROR;
    end case;
    SHOW_VALUE_EXPRESSION (  S.OPERANDS );
    PRINT ( " )" );
  end SHOW_ALL_SET_FUNCTION;

  -- 5.9.1 <value expression>

  procedure PARENTHESIZE_ADDING_OPERANDS
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
  begin
    SHOW_VALUE_EXPRESSION ( S );
    PRINT ( P );
    if S.ACROSS.KIND = OPERATION then
      case S.ACROSS.OPERATION is
        when O_UNARY_MINUS | O_PLUS | O_MINUS =>
          PRINT ( "( " );
          SHOW_VALUE_EXPRESSION ( S.ACROSS );
          PRINT ( " )" );
        when others =>
          SHOW_VALUE_EXPRESSION ( S.ACROSS );
      end case;
    else
      SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end if;
end PARENTHESIZE_ADDING_OPERANDS;

procedure PARENTHESIZE_MULTIPLYING_OPERANDS
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  if S.KIND = OPERATION then
    case S.OPERATION is
      when O_UNARY_MINUS | O_PLUS | O_MINUS =>
        PRINT ( "( " );
        SHOW_VALUE_EXPRESSION ( S );
        PRINT ( " )" );
      when others =>
        SHOW_VALUE_EXPRESSION ( S );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S );
  end if;
  PRINT ( P );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
        PRINT ( "( " );
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
        PRINT ( " )" );
      when others =>
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end if;
end PARENTHESIZE_MULTIPLYING_OPERANDS;

procedure SHOW_VALUE_EXPRESSION ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when NAME =>
      PRINT ( S.NAME.all );
    when OPERATION =>
      case S.OPERATION is
        when O_AVG | O_MAX | O_MIN | O_SUM =>
          SHOW_ALL_SET_FUNCTION ( S );
        when O_COUNT_STAR =>
          PRINT ( "COUNT(*)" );
        when O_UNARY_PLUS =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
        when O_UNARY_MINUS =>
          PRINT ( " - " );
          if S.OPERANDS.KIND = OPERATION then
            case S.OPERANDS.OPERATION is
              when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
                PRINT ( "( " );
                SHOW_VALUE_EXPRESSION ( S.OPERANDS );
                PRINT ( " )" );
              when others => SHOW_VALUE_EXPRESSION ( S.OPERANDS );
            end case;
          else
            SHOW_VALUE_EXPRESSION ( S.OPERANDS );
          end if;
        when O_PLUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " + " );
        when O_MINUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " - " );
        when O_TIMES =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " * " );
        when O_DIVIDE =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " / " );
        when others => raise INTERNAL_ERROR;
      end case;
  end case;
end SHOW_VALUE_EXPRESSION;

-- 5.11.1 <comparison predicate>

procedure SHOW_COMPARISON_PREDICATE
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( P );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_SELEC | O_SELECT_DISTINCT =>
        SHOW_QUERY_SPECIFICATION ( S.ACROSS );
      when others =>
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end if;
end SHOW_COMPARISON_PREDICATE;

-- 5.12.1 <between predicate>

procedure SHOW_BETWEEN_PREDICATE ( S : in SQL_OBJECT ) is
  OPERAND : SQL_OBJECT := S.ACROSS.OPERANDS; -- first operand of AND
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " BETWEEN " );
  SHOW_VALUE_EXPRESSION ( OPERAND );
  PRINT ( " AND " );
  SHOW_VALUE_EXPRESSION ( OPERAND.ACROSS );
end SHOW_BETWEEN_PREDICATE;

-- 5.13.1 <in predicate>

procedure SHOW_IN_PREDICATE
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  PRINT ( P );
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " IN " );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_SELEC | O_SELECT_DISTINCT =>
        SHOW_QUERY_SPECIFICATION ( S.ACROSS );
        return;
      when others =>
        null;
    end case;
  end if;
  PRINT ( "< " ); SHOW_IN_VALUE_LIST ( S.ACROSS ); PRINT ( " >" );
end SHOW_IN_PREDICATE;

-- 5.13.2 <in value list>

procedure SHOW_IN_VALUE_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when OPERATION =>
      if S.OPERATION /= O_OR then
        raise INTERNAL_ERROR;
      end if;
      SHOW_IN_VALUE_LIST ( S.OPERANDS );
      PRINT ( ", " );
      SHOW_IN_VALUE_LIST ( S.OPERANDS.ACROSS );
    when others =>
      raise INTERNAL_ERROR;
  end case;
end SHOW_IN_VALUE_LIST;

-- 5.14.1 <like predicate>

procedure SHOW_LIKE_PREDICATE ( S : in SQL_OBJECT ) is
  P : ACCESS_STRING := S.ACROSS.VALUE.STRING; -- must be of right type
begin
  PRINT ( S.NAME.all ); PRINT ( " = " );
```

```
    for I in P'RANGE loop
      case P(I) is
        when '_'    => P(I) := '?';
        when '%'    => P(I) := '*';
        when others => null;
      end case;
    end loop;
    SHOW_VALUE_SPECIFICATION  ( S.ACROSS );
  end SHOW_LIKE_PREDICATE;

  -- 5.18.1 <search condition>

  procedure PARENTHESIZE_RELATIONAL_OPERATORS
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
    OPERAND : SQL_OBJECT := S.OPERANDS;
  begin
    case OPERAND.OPERATION is -- must be operation
      when O_AND | O_OR =>
        if OPERAND.OPERATION /= S.OPERATION then
          PRINT ( "[ " ); SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( " ]" );
        else
          SHOW_SEARCH_CONDITION ( OPERAND );
        end if;
      when others => SHOW_SEARCH_CONDITION ( OPERAND );
    end case;
    PRINT_LINE; PRINT ( P );
    OPERAND := OPERAND.ACROSS;
    case OPERAND.OPERATION is -- again, must be operation
      when O_AND | O_OR =>
        PRINT ( "[ " ); SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( " ]" );
      when others =>
        SHOW_SEARCH_CONDITION ( OPERAND );
    end case;
  end PARENTHESIZE_RELATIONAL_OPERATORS;

  procedure SHOW_SEARCH_CONDITION ( S : in SQL_OBJECT ) is
  begin
    case S.OPERATION is
      when O_EQ      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " = " );
      when O_NE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " ^= " );
      when O_LT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " < " );
      when O_GT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " > " );
      when O_LE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " <= " );
      when O_GE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " >= " );
      when O_BETWEEN => SHOW_BETWEEN_PREDICATE    ( S.OPERANDS );
      when O_IS_IN   => SHOW_IN_PREDICATE         ( S.OPERANDS , "" );
      when O_NOT_IN  => SHOW_IN_PREDICATE         ( S.OPERANDS , "NOT" );
      when O_LIKE    => SHOW_LIKE_PREDICATE       ( S.OPERANDS );
      when O_AND     => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "AND     " );
      when O_OR      => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "OR      " );
    when O_NOT =>
      PRINT ( "NOT " );
      case S.OPERANDS.OPERATION is -- must be operation
        when O_AND | O_OR =>
          PRINT ( "[ " );
          SHOW_SEARCH_CONDITION ( S.OPERANDS );
          PRINT ( " ]" );
        when others =>
          SHOW_SEARCH_CONDITION ( S.OPERANDS );
      end case;
    when others => raise INTERNAL_ERROR;
  end case;
end SHOW_SEARCH_CONDITION;

-- 5.19.1 <table expression>

procedure SHOW_TABLE_EXPRESSION ( S : in SQL_OBJECT ) is
  CLAUSE : SQL_OBJECT := S.ACROSS;
begin
  PRINT ( "FROM     " ); SHOW_SELECT_LIST ( S );
  if CLAUSE.OPERATION /= O_NULL_OP then -- WHERE must have operation on top
    PRINT_LINE; PRINT ( "WHERE    " ); SHOW_SEARCH_CONDITION ( CLAUSE );
  end if;
  CLAUSE := CLAUSE.ACROSS;
  if CLAUSE.KIND /= OPERATION or else CLAUSE.OPERATION /= O_NULL_OP then
    PRINT_LINE; PRINT ( "GROUP  BY " ); SHOW_SELECT_LIST ( CLAUSE );
  end if;
  CLAUSE := CLAUSE.ACROSS;
  if CLAUSE.OPERATION /= O_NULL_OP then -- same as WHERE
    PRINT_LINE; PRINT ( "HAVING " ); SHOW_SEARCH_CONDITION ( CLAUSE );
  end if;
end SHOW_TABLE_EXPRESSION;

-- 5.25.1 <query specification>

procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT ) is
  CLAUSE : SQL_OBJECT := S.OPERANDS;
begin
  INDENT := INDENT + 7; SET_INDENT ( INDENT );  PRINT_LINE;
  PRINT ( "SELECT " );
  case S.OPERATION is
    when O_SELEC           => null;
    when O_SELECT_DISTINCT => PRINT ( "UNIQUE ");
    when others            => raise INTERNAL_ERROR;
  end case;
  SHOW_SELECT_LIST ( CLAUSE );
  PRINT_LINE;
  SHOW_TABLE_EXPRESSION ( CLAUSE.ACROSS );
  INDENT := INDENT - 7;
  if INDENT >= 0 then
```

```
      PRINT ( " ;" ); SET_INDENT ( INDENT );
    end if;
  end SHOW_QUERY_SPECIFICATION;

  -- 5.25.2 <select list>

  procedure SHOW_SELECT_LIST ( S : in SQL_OBJECT ) is
  begin
    case S.KIND is
      when NAME | VALUE =>
        SHOW_VALUE_EXPRESSION ( S );
      when OPERATION =>
        case S.OPERATION is
          when O_STAR =>
            PRINT ( "*" );
          when O_AMPERSAND =>
            SHOW_SELECT_LIST ( S.OPERANDS );
            PRINT ( ", " );
            SHOW_SELECT_LIST ( S.OPERANDS.ACROSS );
          when others =>
            SHOW_VALUE_EXPRESSION ( S );
        end case;
    end case;
  end SHOW_SELECT_LIST;

  -- 8.3.5 <order by clause>

  procedure SHOW_ORDER_BY_CLAUSE ( S : in SQL_OBJECT ) is
  begin
    case S.KIND is
      when NAME =>
        PRINT ( S.NAME.all );
      when OPERATION =>
        case S.OPERATION is
          when O_AMPERSAND =>
            SHOW_ORDER_BY_CLAUSE ( S.OPERANDS );
            PRINT ( ", " );
            SHOW_ORDER_BY_CLAUSE ( S.OPERANDS.ACROSS );
          when O_ASC =>
            PRINT ( S.OPERANDS.NAME.all );
          when O_DESC =>
            PRINT ( S.OPERANDS.NAME.all & " DESC" );
          when others =>
            raise INTERNAL_ERROR;
        end case;
      when others =>
        raise INTERNAL_ERROR;
    end case;
  end SHOW_ORDER_BY_CLAUSE;
  -- 8.7.3 <insert value list>

  procedure SHOW_INSERT_VALUE_LIST ( S : in SQL_OBJECT ) is
  begin
    case S.KIND is
      when VALUE =>
        SHOW_VALUE_SPECIFICATION ( S );
      when OPERATION =>
        case S.OPERATION is
          when O_AND =>
            SHOW_INSERT_VALUE_LIST ( S.OPERANDS );
            PRINT ( ", " );
          when O_LE =>
            null;
          when others =>
            raise INTERNAL_ERROR;
        end case;
        SHOW_INSERT_VALUE_LIST ( S.OPERANDS.ACROSS );
      when others =>
        raise INTERNAL_ERROR;
    end case;
  end SHOW_INSERT_VALUE_LIST;

  -- 8.11.2 <set clause>

  procedure SHOW_SET_CLAUSES ( S : in SQL_OBJECT ) is
  begin
    case S.OPERATION is -- must be operation
      when O_AND =>
        SHOW_SET_CLAUSES ( S.OPERANDS ); PRINT ( "," ); PRINT_LINE;
        SHOW_SET_CLAUSES ( S.OPERANDS.ACROSS );
      when O_LE =>
        PRINT ( S.OPERANDS.NAME.all & " = " );
        SHOW_VALUE_EXPRESSION ( S.OPERANDS.ACROSS );
      when others =>
        raise INTERNAL_ERROR;
    end case;
  end SHOW_SET_CLAUSES;

-- routine to show a cursor

  procedure SHOW_CURSOR
            ( CURSOR : in CURSOR_NAME ; MESSAGE : in STANDARD.STRING ) is
  begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( MESSAGE ); PRINT_LINE;
    INDENT := -7; SHOW_QUERY_SPECIFICATION ( SQL_OBJECT ( CURSOR.OPERANDS ) );
    if CURSOR.OPERANDS.ACROSS /= null then
      PRINT_LINE; PRINT ( "ORDER  BY ");
      SHOW_ORDER_BY_CLAUSE ( CURSOR.OPERANDS.ACROSS );
    end if;
    PRINT ( " /" ); PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end SHOW_CURSOR;

-- close routine

  procedure CLOSE ( CURSOR : in out CURSOR_NAME ) is
  begin
    SHOW_CURSOR ( CURSOR , "Cursor closed for:");
  end CLOSE;

-- declare cursor routines

  procedure DECLAR
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    CURSOR := new
     SQL_OBJECT_RECORD' ( OPERATION , null , O_DECLAR , CURSOR_FOR );
    CURSOR_FOR.ACROSS := ORDER_BY;
    SHOW_CURSOR ( CURSOR , "Cursor declared for:" );
  end DECLAR;

-- delete routines

  procedure DELETE_FROM
            ( TABLE : in TABLE_NAME;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "DELETE " & TABLE.NAME.all );
    if WHERE /= null then
      INDENT := 0; PRINT_LINE; PRINT ( "WHERE  " );
      SHOW_SEARCH_CONDITION ( WHERE );
    end if;
    PRINT ( " /" ); PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end DELETE_FROM;

-- fetch and into routines

  procedure FETCH ( CURSOR : in out CURSOR_NAME ) is
  begin
    SHOW_CURSOR ( CURSOR , "Fetch performed on:" );
  end FETCH;

  procedure INTEGER_AND_ENUMERATION_INTO ( VAR : out USFR_TYPE ) is
  begin
```

```
    PRINT ( "INTO with integer or enumeration argument" ); PRINT_LINE;
    VAR := USER_TYPE'FIRST; -- ***** FOR TEST PURPOSES
  end INTEGER_AND_ENUMERATION_INTO;

  procedure FLOAT_INTO ( VAR : out USER_TYPE ) is
  begin
    PRINT ( "INTO with float argument" ); PRINT_LINE;
    VAR := USER_TYPE'SMALL; -- ***** FOR TEST PURPOSES
  end FLOAT_INTO;

  procedure UNCONSTRAINED_STRING_INTO
            ( VAR : out USER_TYPE ; LAST : out INDEX_TYPE ) is
  begin
    PRINT ( "INTO with unconstrained string argument" ); PRINT_LINE;
    LAST := INDEX_TYPE'FIRST; -- ***** FOR TEST PURPOSES
  end UNCONSTRAINED_STRING_INTO;

  procedure CONSTRAINED_STRING_INTO
            ( VAR : out USER_TYPE ; LAST : out INDEX_TYPE ) is
  begin
    PRINT ( "INTO with constrained string argument" ); PRINT_LINE;
    LAST := INDEX_TYPE'FIRST; -- ***** FOR TEST PURPOSES
  end CONSTRAINED_STRING_INTO;

-- insert into routines

  procedure INSERT_INTO
            ( TABLE : in TABLE_NAME;
              WHAT  : in INSERT_ITEM ) is
  begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "INSERT INTO " );
    if TABLE.KIND = NAME then
      PRINT ( TABLE.NAME.all );
    else -- must be O_TABLE_COLUMN_LIST
      PRINT ( TABLE.OPERANDS.NAME.all );
      PRINT ( "( " );
      SHOW_SELECT_LIST ( TABLE.OPERANDS.ACROSS );
      PRINT ( " )" );
    end if;
    PRINT ( " :" ); PRINT_LINE;
    case WHAT.OPERATION is -- must be an operation
      when O_SELEC | O_SELECT_DISTINCT =>
        INDENT := -7; SHOW_QUERY_SPECIFICATION ( SQL_OBJECT ( WHAT ) );
      when O_LE | O_AND =>
        PRINT ( "< " );
        SHOW_INSERT_VALUE_LIST ( SQL_OBJECT ( WHAT ) );
        PRINT ( " >" );
      when others =>
        raise INTERNAL_ERROR;
    end case;
    PRINT ( " /" ); PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end INSERT_INTO;

  function VALUES return INSERT_ITEM is
  begin
    return new SQL_OBJECT_RECORD' ( OPERATION , null , O_VALUES , null );
  end VALUES;

-- open routine

  procedure OPEN ( CURSOR : in out CURSOR_NAME ) is
  begin
    SHOW_CURSOR ( CURSOR , "Cursor opened for:");
  end OPEN;

-- select statement routines

  procedure SHOW_SELECT ( S : in SQL_OBJECT ) is
  begin
    BLANK_LINE; INDENT := -7;
    SHOW_QUERY_SPECIFICATION ( S );
    PRINT ( " /" ); PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end SHOW_SELECT;

  procedure SELECT_LIST_SELECT
            ( WHAT     : in WHAT_TYPE;
              FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    SHOW_SELECT
    ( BUILD_SELECT
      ( SELECT_TYPE,
        L_CONVERT ( WHAT ) , FROM , WHERE , GROUP_BY , HAVING ) );
  end SELECT_LIST_SELECT;

  procedure STAR_SELECT
            ( FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    SHOW_SELECT
    ( BUILD_SELECT
      ( SELECT_TYPE,
        new SQL_OBJECT_RECORD' ( OPERATION , null , O_STAR , null ),
        FROM , WHERE , GROUP_BY , HAVING ) );
  end STAR_SELECT;

-- update routines

  procedure UPDATE
           ( TABLE : in TABLE_NAME;
             SET   : in SQL_OBJECT;
             WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "UPDATE " & TABLE.NAME.all );
    PRINT_LINE; PRINT ( "SET " ); SET_INDENT ( 4 ); SHOW_SET_CLAUSES ( SET );
    if WHERE /= null then
      INDENT := 0; SET_INDENT ( 0 ); PRINT_LINE; PRINT ( "WHERE  ");
      SHOW_SEARCH_CONDITION ( WHERE );
    end if;
    PRINT ( " /" ); PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end UPDATE;

end ADA_SQL_FUNCTIONS;
```

10. Package EXAMPLE_DDL

```
package EXAMPLE_TYPES is

  package ADA_SQL is

    type DEPT_LOC_CHARACTER is new CHARACTER;
    type DEPT_LOC_INDEX is range 1..15;
    type EMP_JOB_INDEX is range 1..11; -- to allow "programmer%"
    type EMP_NAME_CHARACTER is new CHARACTER;

    type EMP_NUMBER  is range 1..10_000;
    type EMP_NAME    is array(1..10) of EMP_NAME_CHARACTER;
    type DEPT_CODE   is ( ZERO, ADMIN, ESALES, CSALES, WSALES, MKTING,
                    RSRCH, FIN, COLL );
    type EMP_JOB     is array ( EMP_JOB_INDEX range <> ) of CHARACTER;
    type MONTHLY_PAY is digits 6 range 0.0 .. 9999.99;
    type DEPT_NAME   is array (1..15) of CHARACTER;
    type DEPT_LOC    is array ( DEPT_LOC_INDEX range <> ) of
                      DEPT_LOC_CHARACTER;
    --type ANNUAL_PAY  is digits 7 range 0.0 .  99999.99;
    type ANNUAL_PAY  is digits 6 range 0.0 .. 99999.99;
    --type TAX_AMOUNT  is digits 7 range 0.0 .. 99999.99;
    type TAX_AMOUNT  is digits 6 range 0.0 .. 99999.99;
    type TAX_RATE    is digits 5 range 0.0 .. 0.5;

    --type TAX_COMPUTATION_PRECISION is digits 10 range 0.0 .. 99999.99999;
    type TAX_COMPUTATION_PRECISION is digits 6 range 0.0 .. 99999.99999;

    --type TOTAL_PAY is digits 9 range 0.0 .. 9999999.99; -- for summing up $$
    type TOTAL_PAY is digits 6 range 0.0 .. 9999999.99; -- for summing up $$

    subtype EMP_NUMBER_NOT_NULL_UNIQUE is EMP_NUMBER;
    subtype DEPT_CODE_NOT_NULL_UNIQUE  is DEPT_CODE;

    -- Note the four basic kinds of strings we have defined here:

    --                  +------------------------+------------------------+
    --                  | components of type     | components of          |
    --                  | derived from CHARACTER | type CHARACTER         |
    --  +---------------+------------------------+------------------------+
    --  | unconstrained | DEPT_LOC               | EMP_JOB                |
    --  +---------------+------------------------+------------------------+
    --  | constrained   | EMP_NAME               | DEPT_NAME              |
    --  +---------------+------------------------+------------------------+

  end ADA_SQL;

end EXAMPLE_TYPES;

with EXAMPLE_TYPES;
package EXAMPLE_DDL is
  use EXAMPLE_TYPES.ADA_SQL;

  package ADA_SQL is

    type EMP is
      record
        NUMBER     : EMP_NUMBER_NOT_NULL_UNIQUE;
        NAME       : EMP_NAME;
        DEPT       : DEPT_CODE;
        JOB        : EMP_JOB(1..10);
        MANAGER    : EMP_NUMBER;
        SALARY     : MONTHLY_PAY;
        COMMISSION : MONTHLY_PAY;
      end record;

  -- The UNIFY manual from which the examples were taken (UNIFY Relational
  --  Data Base Management System - Reference Manual, Release 3.2 - Part
  --  Number 7011) provides a good example of why strong typing is important:
  --  they state that the SALARY column contains monthly pay while the
  --  COMMISSION column contains annual commission.  Many of the examples
  --  (like those adding SALARY and COMMISSION ) only make sense, however, if
  --  SALARY and COMMISSION are stated for the same time periods.  But there
  --  are also other examples where SALARY is multiplied by 12 before being
  --  added to COMMISSION, as if SALARY were monthly and COMMISSION were
  --  yearly.  Had strong typing been used, such errors would have been more
  --  difficult to commit.  In our translation of the UNIFY examples to
  --  Ada/SQL, we have typed both SALARY and COMMISSION as if they were
  --  monthly.  This avoids having to do a lot of type conversions for the
  --  example operations.  (Examples of type conversions are still given,
  --  however, since they are required for other operations.)  The UNIFY
  --  examples are translated to the corresponding Ada/SQL, without regard for
  --  whether or not each operation really makes sense in the larger context
  --  of all examples.

    type DEPT is
      record
        CODE     : DEPT_CODE_NOT_NULL_UNIQUE;
        NAME     : DEPT_NAME;
        LOCATION : DEPT_LOC(1..15);
      end record;

    type TAXES is
      record
        MIN_AMOUNT    : ANNUAL_PAY;
        MAX_AMOUNT    : ANNUAL_PAY;
        BASE_TAX      : TAX_AMOUNT;
        MARGINAL_RATE : TAX_RATE;
      end record;

    type CAND is
      record
        NUMBER : EMP_NUMBER_NOT_NULL_UNIQUE;
        NAME   : EMP_NAME;
        DEPT   : DEPT_CODE;
        SALARY : MONTHLY_PAY;
      end record;

  end ADA_SQL;

end EXAMPLE_DDL;

with CURSOR_DEFINITION, DATABASE, EXAMPLE_TYPES;
 use CURSOR_DEFINITION, EXAMPLE_TYPES; -- vary the USE for test purposes
package EXAMPLE_VARIABLES is

  use ADA_SQL;

  -- cursors used

  CURSOR : CURSOR_NAME;
  -- variables to obtain database values

  V_NUMBER             : EMP_NUMBER;     -- variable names do not, of course,
  V_EMP_NAME           : EMP_NAME;       --  have to start with "V_" (see
  V_DEPT               : DEPT_CODE;      --  COUNT_RESULT, for example); we
  V_JOB                : EMP_JOB(1..10); --  just use that convention here to
  V_MANAGER            : EMP_NUMBER;     --  ensure that they are distinct
  V_SALARY             : MONTHLY_PAY;    --  from table and column names
  V_MAX_SALARY         : MONTHLY_PAY;
  V_COMMISSION         : MONTHLY_PAY;
  V_MINIMUM_COMMISSION : MONTHLY_PAY;
  V_DEPT_NAME          : DEPT_NAME;
  V_LOCATION           : DEPT_LOC(1..15);
  V_MIN_AMOUNT         : ANNUAL_PAY;
  V_MAX_AMOUNT         : ANNUAL_PAY;
  V_BASE_TAX           : TAX_AMOUNT;
  V_EXTRA_TAX          : TAX_AMOUNT;
  V_ANNUAL_PAY         : ANNUAL_PAY;
  V_MARGINAL_RATE      : TAX_RATE;
  V_TOTAL_PAY          : TOTAL_PAY;
  V_MGR_NAME           : EMP_NAME;
  V_MGR_SALARY         : MONTHLY_PAY;
  V_MGR_LOCATION       : DEPT_LOC(1..15);
  COUNT_RESULT         : DATABASE.INTG;

  STR_LAST        : INTEGER;
  STR_LAST_2      : INTEGER;
  JOB_LAST        : EMP_JOB_INDEX;
  LOCATION_LAST   : DEPT_LOC_INDEX;
  LOCATION_LAST_2 : DEPT_LOC_INDEX;

end EXAMPLE_VARIABLES;
```

11. Package EXAMPLE_ADA_SQL

```
with ADA_SQL_FUNCTIONS, DATABASE, EXAMPLE_TYPES;
package EXAMPLE_ADA_SQL is

  procedure INITIATE_TEST renames ADA_SQL_FUNCTIONS.INITIATE_TEST;

  -- column and table names

  use ADA_SQL_FUNCTIONS.CONVERT;

  package ADA_SQL is

    package BASE_TAX_NAME is new
```

```
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "BASE_TAX" );
package CANDIDATES_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "CANDIDATES" );
package CODE_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "CODE" );
package COMMISSION_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "COMMISSION" );
package DEPT_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "DEPT" );
package EMP_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP" );
package JOB_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "JOB" );
package LOCATION_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "LOCATION" );
package MARGINAL_RATE_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "MARGINAL_RATE" );
package MAX_AMOUNT_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "MAX_AMOUNT" );
package MIN_AMOUNT_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "MIN_AMOUNT" );
package NAME_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "NAME" );
package NUMBER_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "NUMBER" );
package SALARY_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "SALARY" );
package TAXES_NAME is new ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "TAXES" );

package EXAMPLE_TYPES_INDEX_PACKAGE is

  subtype DEPT_NAME_INDEX is POSITIVE range 1  .. 15;
  subtype EMP_NAME_INDEX is INTEGER range 1 .. 10;

end EXAMPLE_TYPES_INDEX_PACKAGE;

package DATABASE_TYPE_PACKAGE is

  type INTG_TYPE is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;

end DATABASE_TYPE_PACKAGE;

package EXAMPLE_TYPES_TYPE_PACKAGE is

  type ANNUAL_PAY_TYPE  is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type DEPT_CODE_TYPE   is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type DEPT_LOC_TYPE    is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type DEPT_NAME_TYPE   is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type EMP_JOB_TYPE     is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type EMP_NAME_TYPE    is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type EMP_NUMBER_TYPE  is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type MONTHLY_PAY_TYPE is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type TAX_AMOUNT_TYPE  is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type TAX_COMPUTATION_PRECISION_TYPE is new
   ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
  type TAX_RATE_TYPE    is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
```

```
end EXAMPLE_TYPES_TYPE_PACKAGE;

use EXAMPLE_TYPES_TYPE_PACKAGE;

package EXAMPLE_TYPES_NAME_PACKAGE is

  package DEPT_TABLE is

    package CODE_NAME is new
     ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "DEPT.CODE" );
    package LOCATION_NAME is new
     ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "DEPT.LOCATION" );

    function CODE_FUNCTION is new
     CODE_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
    function LOCATION_FUNCTION is new
     LOCATION_NAME.COLUMN_OR_TABLE_NAME (ADA_SQL_FUNCTIONS.SQL_OBJECT );
    function LOCATION_FUNCTION is new
     LOCATION_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE );

    type TYPED_TABLE_TYPE is
      record
        CODE     : EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE;
        LOCATION : EXAMPLF_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE;
      end record;

    TYPED_TABLE :
     constant TYPED_TABLE_TYPE :=
      ( CODE     => CODE_FUNCTION,
        LOCATION => LOCATION_FUNCTION );

    type UNTYPED_TABLE_TYPE is
      record
        LOCATION : ADA_SQL_FUNCTIONS.SQL_OBJECT;
      end record;

    UNTYPED_TABLE :
     constant UNTYPED_TABLE_TYPE :=
      ( LOCATION => LOCATION_FUNCTION );

  end DEPT_TABLE;

  package EMP_TABLE is

    package DEPT_NAME is new
     ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.DEPT" );
    package JOB_NAME is new
     ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.JOB" );
```

```
package MANAGER_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.MANAGER" );
package NAME_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.NAME" );
package NUMBER_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.NUMBER" );
package SALARY_NAME is new
 ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP.SALARY" );

function DEPT_FUNCTION is new
 DEPT_NAME.COLUMN_OR_TABLE_NAME
 ( EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
function JOB_FUNCTION is new
 JOB_NAME.COLUMN_OR_TABLE_NAME
 ( EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE );
function MANAGER_FUNCTION is new
 MANAGER_NAME.COLUMN_OR_TABLE_NAME
 ( EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE );
function NAME_FUNCTION is new
 NAME_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function NUMBER_FUNCTION is new
 NUMBER_NAME.COLUMN_OR_TABLE_NAME
 ( EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE );
function SALARY_FUNCTION is new
 SALARY_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function SALARY_FUNCTION is new
 SALARY_NAME.COLUMN_OR_TABLE_NAME
 ( EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

type TYPED_TABLE_TYPE is
  record
    DEPT    : EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE;
    JOB     : EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE;
    MANAGER : EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE;
    NUMBER  : EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE;
    SALARY  : EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE;
  end record;

TYPED_TABLE :
 constant TYPED_TABLE_TYPE :=
  ( DEPT    => DEPT_FUNCTION,
    JOB     => JOB_FUNCTION,
    MANAGER => MANAGER_FUNCTION,
    NUMBER  => NUMBER_FUNCTION,
    SALARY  => SALARY_FUNCTION );

type UNTYPED_TABLE_TYPE is
  record
    NAME , SALARY : ADA_SQL_FUNCTIONS.SQL_OBJECT;
  end record;
```

```
      UNTYPED_TABLE :
       constant UNTYPED_TABLE_TYPE :=
        ( NAME   => NAME_FUNCTION,
          SALARY => SALARY_FUNCTION );

    end EMP_TABLE;

  end EXAMPLE_TYPES_NAME_PACKAGE;

end ADA_SQL;

use ADA_SQL.DATABASE_TYPE_PACKAGE, ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE;

function BASE_TAX is new
 ADA_SQL.BASE_TAX_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function BASE_TAX is new
 ADA_SQL.BASE_TAX_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_AMOUNT_TYPE );
function CANDIDATES is new
 ADA_SQL.CANDIDATES_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.TABLE_NAME );
function CODE is new
 ADA_SQL.CODE_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function CODE is new
 ADA_SQL.CODE_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
function COMMISSION is new
 ADA_SQL.COMMISSION_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function COMMISSION is new
 ADA_SQL.COMMISSION_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );
function DEPT is new ADA_SQL.DEPT_NAME.TABLE_NAME_WITH_COLUMN_LIST;
function DEPT is new
 ADA_SQL.DEPT_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_NAME );
function DEPT is new
 ADA_SQL.DEPT_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_LIST );
function DEPT is new
 ADA_SQL.DEPT_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function DEPT is new
 ADA_SQL.DEPT_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
function DEPT is new
 ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
 ( ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.DEPT_TABLE.TYPED_TABLE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.DEPT_TABLE.TYPED_TABLE );
function DEPT is new
 ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
 ( ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.DEPT_TABLE.UNTYPED_TABLE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.DEPT_TABLE.UNTYPED_TABLE );
function EMP is new
 ADA_SQL.EMP_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_NAME );
function EMP is new
 ADA_SQL.EMP_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_LIST );
function EMP is new
 ADA_SQL.EMP_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function EMP is new
 ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
 ( ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.EMP_TABLE.TYPED_TABLE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.EMP_TABLE.TYPED_TABLE );
function EMP is new
 ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
 ( ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.EMP_TABLE.UNTYPED_TABLE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_NAME_PACKAGE.EMP_TABLE.UNTYPED_TABLE );
function JOB is new
 ADA_SQL.JOB_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function JOB is new
 ADA_SQL.JOB_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE );
function LOCATION is new
 ADA_SQL.LOCATION_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function LOCATION is new
 ADA_SQL.LOCATION_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE );
function MARGINAL_RATE is new
 ADA_SQL.MARGINAL_RATE_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function MAX_AMOUNT is new
 ADA_SQL.MAX_AMOUNT_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );
function MAX_AMOUNT is new
 ADA_SQL.MAX_AMOUNT_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function MIN_AMOUNT is new
 ADA_SQL.MIN_AMOUNT_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );
function MIN_AMOUNT is new
 ADA_SQL.MIN_AMOUNT_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function NAME is new
 ADA_SQL.NAME_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function NAME is new
 ADA_SQL.NAME_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE );
function NUMBER is new
 ADA_SQL.NUMBER_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
function SALARY is new
 ADA_SQL.SALARY_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
```

```
function SALARY is new
 ADA_SQL.SALARY_NAME.COLUMN_OR_TABLE_NAME
 ( ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );
function TAXES is new
 ADA_SQL.TAXES_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_NAME );
function TAXES is new
 ADA_SQL.TAXES_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.TABLE_LIST );
function TAXES is new
 ADA_SQL.TAXES_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );

-- correlation name packages

package DEPT_CORRELATION is

  generic
    CORRELATION_NAME : in STANDARD.STRING;
  package NAME is

    package ADA_SQL is

      package CODE_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".CODE" );
      package DEPT_TABLE_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "DEPT " & CORRELATION_NAME );
      package LOCATION_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".LOCATION" );

    end ADA_SQL;

    function CODE is new
     ADA_SQL.CODE_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
    function DEPT is new
     ADA_SQL.DEPT_TABLE_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.TABLE_NAME );
    function LOCATION is new
     ADA_SQL.LOCATION_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
    function LOCATION is new
     ADA_SQL.LOCATION_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE );

  end NAME;

end DEPT_CORRELATION;

package EMP_CORRELATION is

  generic
    CORRELATION_NAME : in STANDARD.STRING;
  package NAME is

    package ADA_SQL is

      package DEPT_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".DEPT" );
      package EMP_TABLE_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "EMP " & CORRELATION_NAME );
      package JOB_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".JOB" );
      package NAME_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".NAME" );
      package NUMBER_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".NUMBER" );
      package SALARY_COLUMN_NAME is new
       ADA_SQL_FUNCTIONS.NAME_PACKAGE ( CORRELATION_NAME & ".SALARY" );

    end ADA_SQL;

    function DEPT is new
     ADA_SQL.DEPT_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );
    function EMP is new
     ADA_SQL.EMP_TABLE_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.TABLE_LIST );
    function EMP is new
     ADA_SQL.EMP_TABLE_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.TABLE_NAME );
    function JOB is new
     ADA_SQL.JOB_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE );
    function NAME is new
     ADA_SQL.NAME_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
    function NUMBER is new
     ADA_SQL.NUMBER_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE );
    function SALARY is new
     ADA_SQL.SALARY_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
    function SALARY is new
     ADA_SQL.SALARY_COLUMN_NAME.COLUMN_OR_TABLE_NAME
     ( EXAMPLE_ADA_SQL.ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.
        MONTHLY_PAY_TYPE );

  end NAME;

end EMP_CORRELATION;

-- conversion package
```

```
package CONVERT_TO is

  package EXAMPLE_TYPES is

    function ANNUAL_PAY ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return ADA_SQL_FUNCTIONS.SQL_OBJECT renames CONVERT_R;

    function ANNUAL_PAY ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE
      renames CONVERT_R;

    function TAX_AMOUNT ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return ADA_SQL_FUNCTIONS.SQL_OBJECT renames CONVERT_R;

    function TAX_AMOUNT ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_AMOUNT_TYPE
      renames CONVERT_R;

    function TAX_COMPUTATION_PRECISION ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return
      ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_COMPUTATION_PRECISION_TYPE
       renames CONVERT_R;

    function TOTAL_PAY ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
     return ADA_SQL_FUNCTIONS.SQL_OBJECT renames CONVERT_R;

  end EXAMPLE_TYPES;

end CONVERT_TO;

-- conversion functions

function CONVERT_COMPONENT_TO_CHARACTER
         ( C : EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER ) return CHARACTER;

function CONVERT_COMPONENT_TO_CHARACTER
         ( C : EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER ) return CHARACTER;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.FLOAT_CONVERT ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT
 ( EXAMPLE_TYPES.ADA_SQL.DEPT_CODE );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.DEPT_CODE )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;
```

```
function L_CONVERT is new
 ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_CONVERT
 ( EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_INDEX,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.DEPT_LOC )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
 ( ADA_SQL.EXAMPLE_TYPES_INDEX_PACKAGE.DEPT_NAME_INDEX,
   EXAMPLE_TYPES.ADA_SQL.DEPT_NAME );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.DEPT_NAME )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.FLOAT_CONVERT ( DATABASE.DOUBLE_PRECISION );

function R_CONVERT ( R : DATABASE.DOUBLE_PRECISION )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.UNCONSTRAINED_CHARACTER_STRING_CONVERT
 ( EXAMPLE_TYPES.ADA_SQL.EMP_JOB_INDEX , EXAMPLE_TYPES.ADA_SQL.EMP_JOB );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.EMP_JOB )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_CONVERT
 ( ADA_SQL.EXAMPLE_TYPES_INDEX_PACKAGE.EMP_NAME_INDEX,
   EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER,
   EXAMPLE_TYPES.ADA_SQL.EMP_NAME );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.EMP_NAME )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT ( DATABASE.INTG );

function R_CONVERT ( R : DATABASE.INTG ) return ADA_SQL_FUNCTIONS.SQL_OBJECT
 renames L_CONVERT;

function L_CONVERT is new
 ADA_SQL_FUNCTIONS.FLOAT_CONVERT ( EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY );

function R_CONVERT ( R : EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY )
 return ADA_SQL_FUNCTIONS.SQL_OBJECT renames L_CONVERT;
```

```
function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER ) return CHARACTER;

function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
 return EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER;

function CONVERT_CHARACTER_TO_COMPONENT ( C: CHARACTER )
 return EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER;

-- operators

type STAR_TYPE is ( '*' );

function "&" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AMPERSAND,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "&" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AMPERSAND,
   ADA_SQL_FUNCTIONS.TABLE_LIST,
   ADA_SQL_FUNCTIONS.TABLE_NAME,
   ADA_SQL_FUNCTIONS.TABLE_LIST );

function "&" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AMPERSAND,
   DATABASE.DOUBLE_PRECISION,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "+" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_PLUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_AMOUNT_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_AMOUNT_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "-" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_MINUS,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "-" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_MINUS,
   EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "*" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_TIMES,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_COMPUTATION_PRECISION_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.TAX_COMPUTATION_PRECISION_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_GT,
   ADA_SQL.DATABASE_TYPE_PACKAGE.INTG_TYPE,
   DATABASE.INTG,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_GT,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
      ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_GT,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">=" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_GE,
    ADA_SQL.DATABASE_TYPE_PACKAGE.INTG_TYPE,
    DATABASE.INTG,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">=" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_GE,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ">=" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_GE,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "<" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_LT,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "<=" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_LE,
    ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
    EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
    ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "<=" is new
  ADA_SQL_FUNCTIONS.BINARY_OPERATION
  ( ADA_SQL_FUNCTIONS.O_LE,
    ADA_SQL_FUNCTIONS.INSERT_ITEM,
    EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
    ADA_SQL_FUNCTIONS.INSERT_ITEM );
```

```
function "<=" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_LE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "<=" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_LE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "and" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AND,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );

function "and" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AND,
   ADA_SQL_FUNCTIONS.INSERT_ITEM,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC,
   ADA_SQL_FUNCTIONS.INSERT_ITEM );

function "and" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AND,
   ADA_SQL_FUNCTIONS.INSERT_ITEM,
   EXAMPLE_TYPES.ADA_SQL.DEPT_NAME,
   ADA_SQL_FUNCTIONS.INSERT_ITEM );

function "and" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AND,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function "and" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AND,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function ASC is new
```

```
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_ASC,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function AVG is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AVG,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function AVG is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_AVG,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function BETWEEN is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_BETWEEN,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function BETWEEN is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_BETWEEN,
   EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function BETWEEN is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_BETWEEN,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function COUNT ( STAR : STAR_TYPE ) return ADA_SQL_FUNCTIONS.SQL_OBJECT;

function COUNT ( STAR : STAR_TYPE )
 return ADA_SQL.DATABASE_TYPE_PACKAGE.INTG_TYPE;

function DESC is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_DESC,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function EQ is new
```

```
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_EQ,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function EQ is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_EQ,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function EQ is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_EQ,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function EQ is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_EQ,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function EQ is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_EQ,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function INDICATOR is new
 ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
 ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.ANNUAL_PAY_TYPE );

function IS_IN is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_IS_IN,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function IS_IN is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_IS_IN,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function LIKE is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_LIKE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function LIKE is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_LIKE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
   EXAMPLE_TYPES.ADA_SQL.EMP_JOB,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function LIKE is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_LIKE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE,
   EXAMPLE_TYPES.ADA_SQL.EMP_NAME,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function MAX is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_MAX,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function MAX is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_MAX,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

function MIN is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_MIN,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function NE is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_NE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_LOC_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function NE is new
```

```
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_NE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function NE is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_NE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE EMP_NUMBER_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NUMBER_TYPE,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "not" is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_NOT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "or" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_OR,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function "or" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_OR,
   EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
   EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );

function "or" is new
 ADA_SQL_FUNCTIONS.BINARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_OR,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
   EXAMPLE_TYPES.ADA_SQL.DEPT_CODE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );

function SUM is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_SUM,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function SUM is new
 ADA_SQL_FUNCTIONS.UNARY_OPERATION
 ( ADA_SQL_FUNCTIONS.O_SUM,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

-- SQL functions

procedure CLOSE ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
 renames ADA_SQL_FUNCTIONS.CLOSE;

procedure DECLAR
          ( CURSOR     : in out ADA_SQL_FUNCTIONS.CURSOR_NAME;
            CURSOR_FOR : in     ADA_SQL_FUNCTIONS.SQL_OBJECT;
            ORDER_BY   : in     ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                                 ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
 renames ADA_SQL_FUNCTIONS.DECLAR;

procedure DELETE_FROM
          ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
            WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                     ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
 renames ADA_SQL_FUNCTIONS.DELETE_FROM;

procedure FETCH ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
 renames ADA_SQL_FUNCTIONS.FETCH;

procedure INTO is new
 ADA_SQL_FUNCTIONS.FLOAT_INTO ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY );

procedure INTO is new
 ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INTO
 ( EXAMPLE_TYPES.ADA_SQL.DEPT_CODE );

procedure INTO is new
 ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_INTO
 ( EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_INDEX,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER,
   EXAMPLE_TYPES.ADA_SQL.DEPT_LOC );

procedure INTO is new
 ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INTO
 ( ADA_SQL.EXAMPLE_TYPES_INDEX_PACKAGE.DEPT_NAME_INDEX,
   CHARACTER,
   EXAMPLE_TYPES.ADA_SQL.DEPT_NAME );

procedure INTO is new
 ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_INTO
 ( EXAMPLE_TYPES.ADA_SQL.EMP_JOB_INDEX,
   CHARACTER,
   EXAMPLE_TYPES.ADA_SQL.EMP_JOB );

procedure INTO is new
 ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INTO
```

```
  ( ADA_SQL.EXAMPLE_TYPES_INDEX_PACKAGE.EMP_NAME_INDEX,
    EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER,
    EXAMPLE_TYPES.ADA_SQL.EMP_NAME );

procedure INTO is new
 ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INTO
 ( EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER );

procedure INTO is new
 ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INTO ( DATABASE.INTG );

procedure INTO is new
 ADA_SQL_FUNCTIONS.FLOAT_INTO ( EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY );

procedure INTO is new
 ADA_SQL_FUNCTIONS.FLOAT_INTO ( EXAMPLE_TYPES.ADA_SQL.TAX_AMOUNT );

procedure INTO is new
 ADA_SQL_FUNCTIONS.FLOAT_INTO ( EXAMPLE_TYPES.ADA_SQL.TAX_RATE );

procedure INTO is new
 ADA_SQL_FUNCTIONS.FLOAT_INTO ( EXAMPLE_TYPES.ADA_SQL.TOTAL_PAY );

procedure INSERT_INTO
          ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
            WHAT  : in ADA_SQL_FUNCTIONS.INSERT_ITEM )
 renames ADA_SQL_FUNCTIONS.INSERT_INTO;

function VALUES return ADA_SQL_FUNCTIONS.INSERT_ITEM
 renames ADA_SQL_FUNCTIONS.VALUES;

procedure OPEN ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
 renames ADA_SQL_FUNCTIONS.OPEN;

function SELEC is new
 ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
 ( ADA_SQL_FUNCTIONS.O_SELEC,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.INSERT_ITEM );

function SELEC is new
 ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
 ( ADA_SQL_FUNCTIONS.O_SELEC,
   ADA_SQL_FUNCTIONS.SQL_OBJECT,
   ADA_SQL_FUNCTIONS.SQL_OBJECT );

function SELEC is new
 ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
 ( ADA_SQL_FUNCTIONS.O_SELEC,
   ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.DEPT_CODE_TYPE );

  function SELEC is new
   ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
   ( ADA_SQL_FUNCTIONS.O_SELEC,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_JOB_TYPE );

  function SELEC is new
   ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
   ( ADA_SQL_FUNCTIONS.O_SELEC,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.EMP_NAME_TYPE );

  function SELEC is new
   ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
   ( ADA_SQL_FUNCTIONS.O_SELEC,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE,
     ADA_SQL.EXAMPLE_TYPES_TYPE_PACKAGE.MONTHLY_PAY_TYPE );

  function SELEC
           ( WHAT     : STAR_TYPE;
             FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
             WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
             GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
             HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
   return ADA_SQL_FUNCTIONS.SQL_OBJECT;

  procedure SELEC is new
   ADA_SQL_FUNCTIONS.SELECT_LIST_SELECT
   ( ADA_SQL_FUNCTIONS.O_SELEC,
     ADA_SQL_FUNCTIONS.SQL_OBJECT );

  function SELECT_DISTINCT is new
   ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
   ( ADA_SQL_FUNCTIONS.O_SELECT_DISTINCT,
     ADA_SQL_FUNCTIONS.SQL_OBJECT,
     ADA_SQL_FUNCTIONS.SQL_OBJECT );

  procedure UPDATE ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
                     SET   : in ADA_SQL_FUNCTIONS.SQL_OBJECT;
                     WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                              ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
   renames ADA_SQL_FUNCTIONS.UPDATE;

end EXAMPLE_ADA_SQL;
```

```
package body EXAMPLE_ADA_SQL is

  package body DEPT_CORRELATION is
    package body NAME is
      DUMMY : ADA_SQL_FUNCTIONS.SQL_OBJECT; -- due to VAX bug
    end NAME;
  end DEPT_CORRELATION;

  package body EMP_CORRELATION is
    package body NAME is
      DUMMY : ADA_SQL_FUNCTIONS.SQL_OBJECT; -- due to VAX bug
    end NAME;
  end EMP_CORRELATION;

  function CONVERT_COMPONENT_TO_CHARACTER
           ( C : EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER )
   return CHARACTER is
  begin
    return CHARACTER ( C );
  end CONVERT_COMPONENT_TO_CHARACTER;

  function CONVERT_COMPONENT_TO_CHARACTER
           ( C : EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER )
   return CHARACTER is
  begin
    return CHARACTER ( C );
  end CONVERT_COMPONENT_TO_CHARACTER;

  function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
   return CHARACTER is
  begin
    return C;
  end CONVERT_CHARACTER_TO_COMPONENT;

  function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
   return EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER is
  begin
    return EXAMPLE_TYPES.ADA_SQL.DEPT_LOC_CHARACTER ( C );
  end CONVERT_CHARACTER_TO_COMPONENT;

  function CONVERT_CHARACTER_TO_COMPONENT ( C: CHARACTER )
   return EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER is
  begin
   return EXAMPLE_TYPES.ADA_SQL.EMP_NAME_CHARACTER ( C );
  end CONVERT_CHARACTER_TO_COMPONENT;

  function COUNT_FUNCTION is new
   ADA_SQL_FUNCTIONS.COUNT_STAR ( ADA_SQL_FUNCTIONS.SQL_OBJECT );

  function COUNT_FUNCTION is new
```

```
   ADA_SQL_FUNCTIONS.COUNT_STAR ( ADA_SQL.DATABASE_TYPE_PACKAGE.INTG_TYPE );

  function COUNT ( STAR : STAR_TYPE ) return ADA_SQL_FUNCTIONS.SQL_OBJECT is
  begin
    return COUNT_FUNCTION;
  end COUNT;

  function COUNT ( STAR : STAR_TYPE )
   return ADA_SQL.DATABASE_TYPE_PACKAGE.INTG_TYPE is
  begin
    return COUNT_FUNCTION;
  end COUNT;

  function SELEC_STAR_SUBQUERY is new
   ADA_SQL_FUNCTIONS.STAR_SUBQUERY
   ( ADA_SQL_FUNCTIONS.O_SELEC , ADA_SQL_FUNCTIONS.SQL_OBJECT );

  function SELEC
           ( WHAT     : STAR_TYPE;
             FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
             WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
             GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
             HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
                         ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
   return ADA_SQL_FUNCTIONS.SQL_OBJECT is
  begin
    return SELEC_STAR_SUBQUERY ( FROM , WHERE , GROUP_BY , HAVING );
  end SELEC;

end EXAMPLE_ADA_SQL;
```

12. Package DML_SUBS

```
with TEXT_IO;
use  TEXT_IO;
package DML_SUBS is
  procedure FLOAT_TO_STRING
          (NUM : in FLOAT;
           STR : in out STRING );
  procedure TELL_NUM
           (NUM  : in STRING;
            CMT  : in STRING;
            Q1   : in STRING);
  procedure TELL_NUM_2
```

```
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING);
  procedure TELL_NUM_3
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING);
  procedure TELL_NUM_4
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING;
            Q4  : in STRING);
  procedure TELL_NUM_5
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING;
            Q4  : in STRING;
            Q5  : in STRING);
end DML_SUBS;
package body DML_SUBS is

------------------------------------------------------------------------
--
-- FLOAT_TO_STRING

  procedure FLOAT_TO_STRING
          (NUM : in FLOAT;
           STR : in out STRING ) is

    package CONVERT_FLOAT is new FLOAT_IO (FLOAT);
    OVERFLOW   : STRING (1..10) := "**********";
    II : INTEGER range 1..10 := 1;

  begin
    CONVERT_FLOAT.PUT (STR, NUM, 2, 0);
  exception
    when others => STR(1..10) := OVERFLOW (1..10);
  end FLOAT_TO_STRING;


---------------------------------------------------------------------
--
--  TELL_NUM
```

```
  procedure TELL_NUM
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING) is

  begin
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE ("Example number " & NUM & "  unify page " & CMT );
    PUT_LINE (" ");
    PUT_LINE (Q1);
    PUT_LINE (" ");
    PUT_LINE (" ");
  end TELL_NUM;

--------------------------------------------------------------------
--
--  TELL_NUM_2

  procedure TELL_NUM_2
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING) is

  begin
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE ("Example number " & NUM & "  unify page " & CMT );
    PUT_LINE (" ");
    PUT_LINE (Q1);
    PUT_LINE (Q2);
    PUT_LINE (" ");
    PUT_LINE (" ");
  end TELL_NUM_2;

--------------------------------------------------------------------
--
--  TELL_NUM_3

  procedure TELL_NUM_3
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING) is

  begin
```

```
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE ("Example number " & NUM & "  unify page " & CMT );
    PUT_LINE (" ");
    PUT_LINE (Q1);
    PUT_LINE (Q2);
    PUT_LINE (Q3);
    PUT_LINE (" ");
    PUT_LINE (" ");
  end TELL_NUM_3;

---------------------------------------------------------------------
--
--  TELL_NUM_4

  procedure TELL_NUM_4
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING;
            Q4  : in STRING) is

  begin
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE ("Example number " & NUM & "  unify page " & CMT );
    PUT_LINE (" ");
    PUT_LINE (Q1);
    PUT_LINE (Q2);
    PUT_LINE (Q3);
    PUT_LINE (Q4);
    PUT_LINE (" ");
    PUT_LINE (" ");
  end TELL_NUM_4;

---------------------------------------------------------------------
--
--  TELL_NUM_5

  procedure TELL_NUM_5
           (NUM : in STRING;
            CMT : in STRING;
            Q1  : in STRING;
            Q2  : in STRING;
            Q3  : in STRING;
            Q4  : in STRING;
            Q5  : in STRING) is
  begin
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE (" ");
    PUT_LINE ("Example number " & NUM & "  unify page " & CMT );
    PUT_LINE (" ");
    PUT_LINE (Q1);
    PUT_LINE (Q2);
    PUT_LINE (Q3);
    PUT_LINE (Q4);
    PUT_LINE (Q5);
    PUT_LINE (" ");
    PUT_LINE (" ");
  end TELL_NUM_5;

end DML_SUBS;
5%
```

13. Package EX_1

```
with EXAMPLE_DDL, EXAMPLE_TYPES, EXAMPLE_VARIABLES, DATABASE, EXAMPLE_ADA_SQL,
     TEXT_IO, DML_SUBS;
use  EXAMPLE_VARIABLES, DATABASE, EXAMPLE_ADA_SQL, DML_SUBS;

package EX_1 is

procedure EXAMPLE_1;

end EX_1;

package body EX_1 is

procedure EXAMPLE_1 is
  use EXAMPLE_TYPES.ADA_SQL;

  package MGR      is new EMP_CORRELATION.NAME  ( "MGR" );
  package X        is new EMP_CORRELATION.NAME  ( "X" );
  package MGR_DEPT is new DEPT_CORRELATION.NAME ( "MGR_DEPT" );
  procedure PUT_LINE (ITEM : in STRING ) renames TEXT_IO.PUT_LINE;
  procedure NEW_LINE (SPACING : in TEXT_IO.POSITIVE_COUNT := 1)
            renames TEXT_IO.NEW_LINE;
  procedure SET_COL (TO : in TEXT_IO.POSITIVE_COUNT) renames TEXT_IO.SET_COL;
  procedure PUT (ITEM : in STRING ) renames TEXT_IO.PUT;

  F_FLOAT   : FLOAT := 0.0;
  F_STRING  : STRING (1..10) := (others => ' ');
  T_LEN     : INTEGER := 0;
  T_STRING : STRING (1..100) := (others => ' ');
  DLI      : DEPT_LOC_INDEX := 1;

begin

--
-- 001 from page 6-6
--

  TELL_NUM ("001 ","6-6","select * from emp");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => EMP ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("NUMBER EMP_NAME   DEPT   JOB             " &
            "MANAGER    SALARY    COMMISSION ");
  loop
    FETCH ( CURSOR );
    INTO ( V_NUMBER );
      SET_COL (1);
      PUT (EMP_NUMBER'IMAGE (V_NUMBER));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (8);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_DEPT );
      SET_COL (19);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (26);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_MANAGER );
      SET_COL (42);
      PUT (EMP_NUMBER'IMAGE (V_MANAGER));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (50);
      PUT (F_STRING);
    INTO ( V_COMMISSION );
      F_FLOAT := FLOAT ( V_COMMISSION );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (60);
      PUT (F_STRING);
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 002 from page 6-6
--

  TELL_NUM ("002", "6-6","select * from dept");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => DEPT ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT     DEPT_NAME        LOCATION");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_DEPT_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_DEPT_NAME'FIRST + 1);
      T_STRING (1..T_LEN) := STRING
            (V_DEPT_NAME (V_DEPT_NAME'FIRST .. STR_LAST));
      SET_COL (10);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_LOCATION , LOCATION_LAST );
      T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (27);
      PUT (T_STRING (1..T_LEN));
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;
```

```
  CLOSE ( CURSOR );

--
-- 003 from page 6-7
--

  TELL_NUM ("003","6-7","select * from taxes");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => TAXES ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("MIN_AMOUNT MAX_AMOUNT   BASE_TAX   MARGINAL_RATE");
  loop
    FETCH ( CURSOR );
    INTO ( V_MIN_AMOUNT );
      F_FLOAT := FLOAT ( V_MIN_AMOUNT );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (1);
      PUT (F_STRING);
    INTO ( V_MAX_AMOUNT );
      F_FLOAT := FLOAT ( V_MAX_AMOUNT);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (12);
      PUT (F_STRING);
    INTO ( V_BASE_TAX );
      F_FLOAT := FLOAT ( V_BASE_TAX);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (23);
      PUT (F_STRING);
    INTO ( V_MARGINAL_RATE );
      F_FLOAT := FLOAT ( V_MARGINAL_RATE);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (34);
      PUT (F_STRING)·
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 004 from page 6-8
--
```

```
  TELL_NUM ("004","6-8","select number, job, name, salary from emp");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( NUMBER & JOB & NAME & SALARY,
    FROM => EMP ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("NUMBER JOB             EMP_NAME            SALARY");
  loop
    FETCH ( CURSOR );
    INTO ( V_NUMBER );
      SET_COL (1);
      PUT (EMP_NUMBER'IMAGE (V_NUMBER));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (8);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (24);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (40);
      PUT (F_STRING);
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 005 from page 6-9
--

  TELL_NUM ("005","6-9","select name, location from dept where code = FIN");

begin
  SELEC  ( NAME & LOCATION,
  FROM  => DEPT,
  WHERE => EQ ( CODE , FIN ) ); -- ***** NOTE USE OF ENUMERATION TYPE
  PUT_LINE ("DEPT_NAME       LOCATION");
  INTO ( V_DEPT_NAME , STR_LAST );
    T_LEN := INTEGER (STR_LAST - V_DEPT_NAME'FIRST + 1);
    T_STRING (1..T_LEN) := STRING
            (V_DEPT_NAME (V_DEPT_NAME'FIRST .. STR_LAST));
    SET_COL (1);
    PUT (T_STRING (1..T_LEN));
  INTO ( V_LOCATION , LOCATION_LAST );
    T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
    for I in 1..T_LEN loop
      DLI := DEPT_LOC_INDEX (I);
      T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
    end loop;
    SET_COL (17);
    PUT (T_STRING (1..T_LEN));
    NEW_LINE;
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;

--
-- 006 from page 6-10
--

  TELL_NUM ("006","6-10",
      "select name, location from dept where location = 'Dallas%'");

  DECLAR ( CURSOR , CURSOR_FOR =>                -- note that Ada/SQL pattern
    SELEC  ( NAME & LOCATION,                    --  matching comparisons are
    FROM  => DEPT,                               --  done with LIKE.  Also note
    WHERE => LIKE ( LOCATION , "Dallas%" ) ) ); --  Ada/SQL %, not UNIFY *

  OPEN ( CURSOR );

begin
    PUT_LINE ("DEPT_NAME       LOCATION");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_DEPT_NAME'FIRST + 1);
      T_STRING (1..T_LEN) := STRING
            (V_DEPT_NAME (V_DEPT_NAME'FIRST .. STR_LAST));
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_LOCATION , LOCATION_LAST );
      T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (17);
      PUT (T_STRING (1..T_LEN));
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 007 from page 6-10
--
--
-- example on page 6-10 is not legal ANSI SQL -- range of characters is not
--  supported for string pattern matching -- following query skipped
-- select Name, Job
-- from   emp
-- where  Name = '[A-M]*' /

  TELL_NUM ("007","6-10","select name, job from emp where name = '[A-M]*'");
  PUT_LINE ("This example is not legal ANSI SQL -- range of characters is not");
  PUT_LINE ("supported for string pattern matching");
  PUT_LINE ("This example is not executed here");

--
-- 008 from page 6-11
--

  TELL_NUM ("008","6-11 ",
          "select name, job from emp where name = '_____     '");

  DECLAR ( CURSOR , CURSOR_FOR =>              -- again note LIKE for pattern
    SELEC  ( NAME & JOB,                       --  matching comparisons.  Also
    FROM  => EMP,                              --  Ada/SQL underscore instead
    WHERE => LIKE ( NAME , "_____     " ) ) ); --  of UNIFY question mark

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME        JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (17);
      PUT (T_STRING (1..T_LEN));
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;
  CLOSE ( CURSOR );

--
-- 009 from page 6-11
--

  TELL_NUM ("009","6-11 ",
   "select name, job, salary, commission from emp where commission > salary");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & SALARY & COMMISSION,
    FROM  => EMP,
    WHERE => COMMISSION > SALARY ) );

  OPEN ( CURSOR );

begin
    PUT_LINE ("EMP_NAME        JOB                 SALARY COMMISSION");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (17);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (33);
      PUT (F_STRING);
    INTO ( V_COMMISSION );
      F_FLOAT := FLOAT ( V_COMMISSION );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (44);
      PUT (F_STRING);
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 010 from page 6-12
--

  TELL_NUM_2 ("010","6-12",
          "select name, job, salary, dept from emp where dept = ADMIN and",
          "[job = 'clerk%' or salary < = 1200]");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & SALARY & DEPT, -- ***** TESTING NOTE: DOES UNIFY
    FROM  => EMP,                        --  REQUIRE < (space) = AS IN
    WHERE => EQ ( DEPT , ADMIN )         --  EXAMPLE?
    AND      ( LIKE ( JOB , "clerk%" ) or SALARY <= 1200.0 ) ) );

  OPEN ( CURSOR );

begin
    PUT_LINE ("EMP_NAME         JOB                  SALARY  DEPT");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (35);
      PUT (F_STRING);
    INTO ( V_DEPT );
      SET_COL (47);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 011 from page 6-12
--

  TELL_NUM ("011 ","6-12",
  "select name, salary, job from emp where salary between 1500.00 and 2000.00");

  DECLAR ( CURSOR , CURSOR_FOR =>                         -- ***** TESTING
    SELEC  ( NAME & SALARY & JOB,                         --  NOTE: ALSO TRY
    FROM  => EMP,                                         --  TYPE QUALIFYING
    WHERE => BETWEEN ( SALARY , 1500.0 and 2000.0 ) ) ); --  NUMBERS

  OPEN ( CURSOR );

begin
    PUT_LINE ("EMP_NAME            SALARY JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (17);
      PUT (F_STRING);
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (28);
      PUT (T_STRING (1..T_LEN));
  end loop;
  NEW_LINE;
exception
  when NOT_FOUND_ERROR => null;
end;
```

```
  CLOSE ( CURSOR );

--
-- 012 from page 6-13
--

  TELL_NUM_2 ("012","6-13","select dept, name, job, salary from emp",
          "where dept = RSRCH and job ^= 'engineer%'");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( DEPT & NAME & JOB & SALARY,
    FROM  => EMP,                                   -- ***** TESTING NOTE: SEE
    WHERE => EQ ( DEPT , RSRCH )                    --  IF UNIFY HANDLES NON-
    AND      not LIKE ( JOB , "engineer%" ) ) ); --  LEADING NOT

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT      EMP_NAME         JOB                    SALARY");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (11);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (28);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (45);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 013 from page 6-13
```

```
--

  TELL_NUM_2 ("013","6-13","select name, job, salary from emp where",
              "not [job = 'salesman%' or salary > = 2000.00]");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & SALARY,
    FROM  => EMP,
    WHERE => not ( LIKE ( JOB , "salesman%" ) or SALARY >= 2000.0 ) ) );

  OPEN ( CURSOR );

begin
    PUT_LINE ("EMP_NAME          JOB                   SALARY");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (19);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (37);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 014 from page 6-14
--

  TELL_NUM ("014","6-14",
  "select name, job, dept from emp where dept is in <ESALES, CSALES, WSALES>");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & DEPT,
    FROM  => EMP,
    WHERE => IS_IN ( DEPT , ESALES or CSALES or WSALES ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         JOB              DEPT");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1),
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_DEPT );
      SET_COL (35);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 015 from page 6-15
--
-- example on page 6-15 is not legal ANSI SQL -- cannot build literal tuples
-- select Name, Job, Salary, Dept-No
-- from   emp
-- where  < Job, Dept_No > is in ( < 'clerk*'     , 10 >,
--                                 < 'programmer*', 60 > ) /

  TELL_NUM_2 ("015","6-15",
       "select name, job, salary, dept from emp where <job, dept> is in",
       "( < 'clerk%', ADMIN >, < 'programmer%', RSRCH > )");
  PUT_LINE (
    "This example is not legal ANSI SQL -- cannot build literal tuples");
  PUT_LINE ("This example is not executed here");

--
-- 016 from page 6-16
--
```

```
  TELL_NUM ("016","6-16","select unique job from emp");

  DECLAR ( CURSOR , CURSOR_FOR => -- note Ada/SQL SELECT_DISTINCT vs. UNIFY's
    SELECT_DISTINCT ( JOB,        --  SELECT UNIQUE
    FROM => EMP ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 017 from page 6-16
--

  TELL_NUM_2 ("017","6-16","select dept, job from emp where dept is in",
          " < ADMIN, ESALES, CSALES > or salary > 2000.00");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( DEPT & JOB,
    FROM  => EMP,
    WHERE => IS_IN ( DEPT , ADMIN or ESALES or CSALES )
    OR       SALARY > 2000.0 ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT            JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (17);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 018 from page 6-17
--

  TELL_NUM_2 ("018","6-17","select unique dept, job from emp where dept is in",
      "< ADMIN, ESALES, CSALES > or salary > 2000.00");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELECT_DISTINCT ( DEPT & JOB,
    FROM  => EMP,
    WHERE => IS_IN ( DEPT , ADMIN or ESALES or CSALES )
    OR       SALARY > 2000.0 ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT              JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (19);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 019 from page 6-18
--

  TELL_NUM_2 ("019","6-18",
       "select name, job, salary + commission from emp where",
       "dept is in < ESALES, CSALES, WSALES >");
  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & ( SALARY + COMMISSION ), -- note parentheses
    FROM  => EMP,                                  --  required
    WHERE => IS_IN ( DEPT , ESALES or CSALES or WSALES ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         JOB                SALARY+COMMISSION");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (35);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 020 from page 6-18
--

  TELL_NUM_2 ("020","6-18",
       "select name, salary, commission, (salary * 0.5 + 100.00) from emp",
       "where commission < salary * 0.5 + 100.00 and job = 'salesman%'");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & SALARY & COMMISSION & ( SALARY * 0.5 + 100.0 ),
    FROM  => EMP,
    WHERE => COMMISSION < SALARY * 0.5 + 100.0
    AND      LIKE ( JOB , "salesman%" ) ) );

  OPEN ( CURSOR );
```

```
begin
  PUT_LINE ("EMP_NAME             SALARY  COMMISSION  SALARY*.5+100");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (18);
      PUT (F_STRING);
    INTO ( V_COMMISSION );
      F_FLOAT := FLOAT ( V_COMMISSION );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (30);
      PUT (F_STRING);
    INTO ( V_MINIMUM_COMMISSION );
      F_FLOAT := FLOAT ( V_MINIMUM_COMMISSION);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (42);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 021 from page 6-19
--

-- as the below example shows, there are some disadvantages to strong
--  typing, and the typing on the literals is not really correct anyway, but
--  mostly for convenience
-- note that arithmetic-type operations not including a database value will
--  be processed totally by Ada, so individual operands will not go to the
--  database.  (These operations may not be redefined!)  To get values to
--  the database here, INDICATOR is used to build a "database" value from
--  the literal.  Note that a literal parameter to INDICATOR may require
--  explicit type specification to establish typing.

  TELL_NUM_3 ("021 ","6-19", "select (3000 * 12) + 5000, base_tax,",
     "((( 3000 * 12 ) + 5000) - min_amount ) * marginal_rate",
     "from taxes where ( 3000 * 12 ) + 5000 between min_amount and max_amount");
  PUT_LINE (
      "in this example the arithemetic values are passed to the database");
  PUT_LINE (" ");
  PUT_LINE (" ");

begin
  PUT_LINE ("3000*12+5000   BASE_TAX " &
       "(((3000*12)+5000)-MIN_AMOUNT)*MARGINAL_RATE");
  SELEC  ( ( INDICATOR ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY'(3000.0) ) * 12.0 +
              5000.0 ) &
           BASE_TAX &
           CONVERT_TO.EXAMPLE_TYPES.TAX_AMOUNT
           ( CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
             ( INDICATOR ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY'(3000.0) ) * 12.0
                + 5000.0 - MIN_AMOUNT ) *
             CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
             ( MARGINAL_RATE ) ),
  FROM  => TAXES,
  WHERE => BETWEEN
           ( INDICATOR ( EXAMPLE_TYPES.ADA_SQL.ANNUAL_PAY'(3000.0) ) * 12.0 +
              5000.0,
             MIN_AMOUNT and MAX_AMOUNT ) );
  INTO ( V_ANNUAL_PAY );
    F_FLOAT := FLOAT ( V_ANNUAL_PAY);
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (1);
    PUT (F_STRING);
  INTO ( V_BASE_TAX );
    F_FLOAT := FLOAT ( V_BASE_TAX);
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (14);
    PUT (F_STRING);
  INTO ( V_EXTRA_TAX );
    F_FLOAT := FLOAT ( V_EXTRA_TAX );
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (30);
    PUT (F_STRING);
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;

--
-- 022 derived from example on page 6-19
--

-- here is the above example redone with the literal math performed by Ada
-- Note the difference in what goes to the database.
-- Also note that there is still a disadvantage to strong typing when
--  operating on expressions that make sense, although they involve
--  different types.

  TELL_NUM_3 ("022","6-19 modified",
     "select (3000 * 12) + 5000, base_tax,",
     "((( 3000 * 12 ) + 5000) - min_amount ) * marginal_rate",
     "from taxes where ( 3000 * 12 ) + 5000 between min_amount and max_amount");
  PUT_LINE (
    "this is the above example redone with the literal math performed by Ada");
  PUT_LINE (" ");
  PUT_LINE (" ");

begin
  PUT_LINE ("3000*12+5000   BASE_TAX " &
       "(((3000*12)+5000)-MIN_AMOUNT)*MARGINAL_RATE"),
  SELEC  ( ( 3000.0 * 12.0 + 5000.0 ) &
           BASE_TAX &
           CONVERT_TO.EXAMPLE_TYPES.TAX_AMOUNT
           ( CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
             ( 3000.0 * 12.0 + 5000.0 - MIN_AMOUNT ) *
             CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
             ( MARGINAL_RATE ) ),
  FROM  => TAXES,
  WHERE => BETWEEN ( 3000.0 * 12.0 + 5000.0 , MIN_AMOUNT and MAX_AMOUNT ) );
  INTO ( V_ANNUAL_PAY );
    F_FLOAT := FLOAT ( V_ANNUAL_PAY);
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (1);
    PUT (F_STRING);
  INTO ( V_BASE_TAX );
    F_FLOAT := FLOAT ( V_BASE_TAX);
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (14);
    PUT (F_STRING);
  INTO ( V_EXTRA_TAX );
    F_FLOAT := FLOAT ( V_EXTRA_TAX );
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (30);
    PUT (F_STRING);
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;

--
-- 023 from page 6-20
--

  TELL_NUM ("023","6-20","select number, name, job from emp order by number");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( NUMBER & NAME & JOB,
    FROM => EMP ),
  ORDER_BY => NUMBER );

  OPEN ( CURSOR );

begin
  PUT_LINE ("NUMBER  EMP_NAME         JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_NUMBER );
      SET_COL (1);
      PUT (EMP_NUMBER'IMAGE (V_NUMBER));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (9);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (26);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 024 from page 6-21
--

  TELL_NUM ("024","6-21 ",
       "select dept, name, job from emp order by dept desc, name asc");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( DEPT & NAME & JOB,
    FROM => EMP ),
  ORDER_BY => DESC ( DEPT ) & ASC ( NAME ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT             EMP_NAME         JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (35);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 025 from page 6-22
--

  TELL_NUM ("025","6-22","select count (*) from emp where dept = ADMIN");

begin
  PUT_LINE ("COUNT (*)");
  SELEC  ( COUNT ('*'),
  FROM  => EMP,
  WHERE => EQ ( DEPT , ADMIN ) );
  INTO ( COUNT_RESULT );
    SET_COL (1);
    PUT (DATABASE.INTG'IMAGE (COUNT_RESULT));
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;
end EXAMPLE_1;

end EX_1;
```

14. Package EX_2

```
with EXAMPLE_DDL, EXAMPLE_TYPES, EXAMPLE_VARIABLES, DATABASE, EXAMPLE_ADA_SQL,
     TEXT_IO, DML_SUBS;
```

```
use                                 EXAMPLE_VARIABLES, DATABASE, EXAMPLE_ADA_SQL,
              DML_SUBS;
package EX_2 is

procedure EXAMPLE_2;

end EX_2;

package body EX_2 is

procedure EXAMPLE_2 is
  use EXAMPLE_TYPES.ADA_SQL;

  package MGR      is new EMP_CORRELATION.NAME  ( "MGR" );
  package X        is new EMP_CORRELATION.NAME  ( "X" );
  package MGR_DEPT is new DEPT_CORRELATION.NAME ( "MGR_DEPT" );
  procedure PUT_LINE (ITEM : in STRING ) renames TEXT_IO.PUT_LINE;
  procedure NEW_LINE (SPACING : in TEXT_IO.POSITIVE_COUNT := 1)
            renames TEXT_IO.NEW_LINE;
  procedure SET_COL (TC : in TEXT_IO.POSITIVE_COUNT) renames TEXT_IO.SET_COL;
  procedure PUT (ITEM : in STRING ) renames TEXT_IO.PUT;

  F_FLOAT  : FLOAT := 0.0;
  F_STRING : STRING (1..10) := (others => ' ');
  T_LEN    : INTEGER := 0;
  T_STRING : STRING (1..100) := (others => ' ');
  DLI      : DEPT_LOC_INDEX := 1;


begin


--
-- 026 from page 6-22
--

  TELL_NUM ("026","6-22","select min (salary), max (salary) from emp");

begin
  PUT_LINE ("MIN(SALARY)     MAX(SALARY)");
  SELEC ( MIN ( SALARY ) & MAX ( SALARY ),
  FROM => EMP );
  INTO ( V_SALARY );
    F_FLOAT := FLOAT ( V_SALARY );
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (1);
    PUT (F_STRING);
  INTO ( V_MAX_SALARY );
    F_FLOAT := FLOAT ( V_MAX_SALARY  );
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (17);
    PUT (F_STRING);
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;

--
-- 027 from 6-23
--
-- example on page 6-23 is not legal ANSI SQL -- when selecting without
--  groups, if one <value expression> in the <select list> includes a <set
--  function specification>, then all <column specification>s in the <select
--  list> must be contained within <set function specification>s

-- select Job, avg ( Salary + Commission )
-- from   emp
-- where  Job = 'salesman*' /

  TELL_NUM ("027","6-23",
    "select job, avg (salary + commission) from emp where job = 'salesman%'");
  PUT_LINE ("This example is not legal ANSI SQL -- when selecting without");
  PUT_LINE (
   "groups, if one <value expression> in the <select list> includes a <set");
  PUT_LINE (
   "function specification>, then all <column specification>s in the <select");
  PUT_LINE ("list> must be contained within <set function specification>s");
  PUT_LINE ("This example is not executed here");

--
-- 028 from 6-23
--

  TELL_NUM_2 ("028","6-23",
     "select sum (( salary * 12 ) + commission) from emp",
     "where job = 'engineer%' or job = 'programmer%'");

begin
  PUT_LINE ("SUM(SALARY+COMMISSION)");
  SELEC  ( CONVERT_TO.EXAMPLE_TYPES.TOTAL_PAY       -- type conversion needed
           ( SUM ( SALARY * 12.0 + COMMISSION ) ), --  to allow for expanded
  FROM  => EMP,                                    --  range
  WHERE => LIKE ( JOB , "engineer%" ) or LIKE ( JOB , "programmer%" ) );
  INTO ( V_TOTAL_PAY );
    F_FLOAT := FLOAT ( V_TOTAL_PAY );
    FLOAT_TO_STRING (F_FLOAT, F_STRING);
    SET_COL (1);
    PUT (F_STRING);
exception
  when NOT_FOUND_ERROR => PUT_LINE ("Selec not found");
  when UNIQUE_ERROR => PUT_LINE ("Selec not unique");
end;

-- types are not precisely correct in above -- should convert both SALARY
--  and COMMISSION before computation to allow for correct ranges, but no
--  existing databases range check, so we took the easy way out

--
-- 029 from page 6-24
--

  TELL_NUM ("029","6-24",
  "select dept, count (*), sum ( salary + commission ) from emp group by dept");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC     ( DEPT & COUNT('*') &
                 CONVERT_TO.EXAMPLE_TYPES.TOTAL_PAY -- see type conversion
                 ( SUM ( SALARY + COMMISSION ) ),   --  comments above
    FROM     => EMP,
    GROUP_BY => DEPT ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT   COUNT(*)         SUM(SALARY+COMMISSION)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( COUNT_RESULT );
      SET_COL (18);
      PUT (DATABASE.INTG'IMAGE (COUNT_RESULT));
    INTO ( V_TOTAL_PAY );
      F_FLOAT := FLOAT ( V_TOTAL_PAY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (30);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 030 from page 6-24
--

  TELL_NUM_2 ("030","6-24",
     "select dept, job, count (*) avg ( salary ) from emp",
     "where dept is in <ADMIN, ESALES, CSALES > group by dept, job");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC     ( DEPT & JOB & COUNT('*') & AVG ( SALARY ),
    FROM     => EMP,
    WHERE    => IS_IN ( DEPT , ADMIN or ESALES or CSALES ),
    GROUP_BY => DEPT & JOB ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT        JOB              COUNT(*)       AVG(SALARY)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (13);
      PUT (T_STRING (1..T_LEN));
    INTO ( COUNT_RESULT );
      SET_COL (30);
      PUT (DATABASE.INTG'IMAGE (COUNT_RESULT));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (45);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 031 from page 6-25
--

  TELL_NUM_2 ("031 ","6-25",
       "select dept, count (*), avg (( salary * 12 ) + commission)",
       "from emp where job = 'salesman%' group by dept");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC     ( DEPT & COUNT('*') &
                 CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY      -- see comments above
                 ( AVG ( SALARY * 12.0 + COMMISSION ) ), -- on type conversion
    FROM     => EMP,
    WHERE    => LIKE ( JOB , "salesman%" ),
    GROUP_BY => DEPT ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT        COUNT(*)      AVG((SALARY*12)+COMMISSION)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( COUNT_RESULT );
      SET_COL (13);
      PUT (DATABASE.INTG'IMAGE (COUNT_RESULT));
    INTO ( V_ANNUAL_PAY );
      F_FLOAT := FLOAT ( V_ANNUAL_PAY);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (27);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 032 from page 6-25
--
-- example on page 6-25 is not legal ANSI SQL -- one <set function
--  specification> cannot be included within another one
-- select avg ( count(*) )
-- from   emp
-- group  by Dept_No /

  TELL_NUM ("032","6-25","select avg (count (*)) from emp group by dept");
  PUT_LINE ("This example is not legal ANSI SQL -- one <set function");
  PUT_LINE ("specification> cannot be included within another one");
  PUT_LINE ("This example is not executed here");

--
-- 033 from page 6-26
--
-- likewise for example on page 6-26
-- select max ( avg ( Salary ) )
-- from   emp
-- where  Job ^= 'president*'
-- group  by Job /
```

```
  TELL_NUM ("033","6-26","select max ( avg ( salary ) ) " &
       "from emp where job ^= 'president%' group by job");
  PUT_LINE ("This example is not legal ANSI SQL -- one <set function");
  PUT_LINE ("specification> cannot be included within another one");
  PUT_LINE ("This example is not executed here");

--
-- 034 from page 6-27
--

  TELL_NUM_2 ("034","6-27",
       "select name, job from emp where salary + commission =",
       "       select max ( salary + commission) from emp");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB,
    FROM   => EMP,
    WHERE => EQ ( SALARY + COMMISSION,
      SELEC ( MAX ( SALARY + COMMISSION ),
      FROM => EMP ) ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         JOB");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 035 from page 6-27
--
-- example on page 6-27 is not legal ANSI SQL -- subqueries can only compare
--  one value at a time, not tuples of values
-- select Name, Job, Salary
-- from   emp
-- where  Dept_No ^= 10
--        and < Job, Salary > is in
--            select Job, Salary
--            from   emp
--            where  Dept_No = 10 /

  TELL_NUM_3 ("035","6-27", "select name, job, salary from emp",
     "       where dept ^= ADMIN and < job, salary > is in",
     "             select job, salary from emp where dept = ADMIN");
  PUT_LINE (
       "This example is not legal ANSI SQL -- subqueries can only compare");
  PUT_LINE ("one value at a time, not tuples of values");
  PUT_LINE ("This example is not executed here");

--
-- 036 from page 6-28
--

  TELL_NUM_4 ("036","6-28","select dept, name, job, salary + commission " &
                   "from emp where salary + commission =",
     "  select max ( salary + commission ) from emp where name ^=",
     "    select name from emp where salary + commission =",
     "      select max ( salary + commission ) from emp");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( DEPT & NAME & JOB & ( SALARY + COMMISSION ),
    FROM  => EMP,
    WHERE => EQ ( SALARY + COMMISSION,
      SELEC  ( MAX ( SALARY + COMMISSION ),
      FROM  => EMP,
      WHERE => NE ( NAME,
        SELEC  ( NAME,
        FROM  => EMP,
        WHERE => EQ ( SALARY + COMMISSION,
          SELEC ( MAX ( SALARY + COMMISSION ),
          FROM => EMP ) ) ) ) ) ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT        EMP_NAME        JOB              SALARY+COMMISSION");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_EMP_NAME , STR_LAST );
```

```
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (13);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOb FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (30);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (47);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 037 from page 6-29
--

  TELL_NUM_4 ("037","6-29",
  "select dept, name, job, salary + commission, commission from emp",
  "   where salary + commission = select max ( salary + commission ) from emp",
  "   where job = 'salesman%';  or [ job = 'salesman%' and dept = ESALES ]",
  "   order by commission desc, salary desc");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC (  DEPT & NAME & JOB & ( SALARY + COMMISSION ) & COMMISSION,
    FROM  => EMP,
    WHERE => EQ ( SALARY + COMMISSION,      -- function parentheses group
      SELEC  ( MAX ( SALARY + COMMISSION ), --  Ada/SQL subqueries; something
      FROM  => EMP,                         --  like UNIFY ; is not needed
      WHERE => LIKE ( JOB , "salesman%" ) ) )
    OR          ( LIKE ( JOB , "salesman%" ) and EQ ( DEPT , ESALES ) ) ),
  ORDER_BY => DESC ( COMMISSION ) & DESC ( SALARY ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT             EMP_NAME         JOB              " &
          "    SALARY  COMMISSION");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (35);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (52);
      PUT (F_STRING);
    INTO ( V_COMMISSION );
      F_FLOAT := FLOAT ( V_COMMISSION );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (64);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 038 from page 6-30
--

  TELL_NUM ("038","6-30","select dept avg (salary) from emp group by " &
       "dept having avg (salary) > 2000.00");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC      ( DEPT & AVG ( SALARY ),
    FROM      => EMP,
    GROUP_BY => DEPT,
    HAVING    => AVG ( SALARY ) > 2000.0 ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT              AVG(SALARY)");
  loop
```

```
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (18);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 039 from page 6-30
--

  TELL_NUM_2 ("039","6-30",
       "select dept, count (*) from emp where job = 'salesman%' ",
       "group by dept having count (*) > 2");
  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC     ( DEPT & COUNT('*'),
    FROM     => EMP,
    WHERE    => LIKE ( JOB , "salesman%" ),
    GROUP_BY => DEPT,
    HAVING   => COUNT('*') > 2 ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT             COUNT(*)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( COUNT_RESULT );
      SET_COL (18);
      PUT (DATABASE.INTG'IMAGE (COUNT_RESULT));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
```

```
-- 040 from page 6-31
--

  TELL_NUM_3 ("040","6-31 ","select dept, avg (salary), avg (commission), " &
                      "avg (salary * 0.5 + 100) from emp",
     "     where job = 'salesman%' group by dept",
     "     having avg (commission) >= avg (salary * 0.5 + 100)");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC      ( DEPT & AVG ( SALARY ) & AVG ( COMMISSION ) &
                 AVG ( SALARY * 0.5 + 100.0 ),
    FROM     => EMP,
    WHERE    => LIKE ( JOB , "salesman%" ),
    GROUP_BY => DEPT,
    HAVING   => AVG ( COMMISSION ) >= AVG ( SALARY * 0.5 + 100.0 ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT        AVG(SALARY)      AVG(COMMISSION)   " &
       "AVG((SALARY*.5)+100)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (14);
      PUT (F_STRING);
    INTO ( V_COMMISSION );
      F_FLOAT := FLOAT ( V_COMMISSION );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (32);
      PUT (F_STRING);
    INTO ( V_MINIMUM_COMMISSION );
      F_FLOAT := FLOAT ( V_MINIMUM_COMMISSION);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (53);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 041 from page 6-31
```

```
--

  TELL_NUM_2 ("041 ","6-31 ",
  "select dept, avg (salary) from emp group by dept having avg (salary) <",
  "       select avg (salary) from emp");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC     ( DEPT & AVG ( SALARY ),
    FROM     => EMP,
    GROUP_BY => DEPT,
    HAVING   => AVG ( SALARY ) <
      SELEC ( AVG ( SALARY ),
      FROM => EMP ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("DEPT      AVG(SALARY)");
  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT );
      SET_COL (1);
      PUT (DEPT_CODE'IMAGE (V_DEPT));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (18);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 042 from page 6-32
--
-- example on page 6-32 is not legal ANSI SQL -- cannot nest set functions
-- select Name, Job, Salary
-- from   emp
-- where  Dept_No =
--   select Dept_No
--   from   emp
--   group  by Dept_No
--   having avg(Salary) =
--     select max(avg(Salary))
--     from   emp
--     group  by Dept_No /
```

```
  TELL_NUM_3 ("042","6-32","select name, job, salary from emp where dept =",
     "       select dept from emp group by dept having avg (salary) =",
     "              select max (avg (Salary)) from emp group by dept");
  PUT_LINE ("This example is not legal ANSI SQL -- cannot nest set functions");
  PUT_LINE ("This example is not executed here");

--
-- 043 from page 6-34
--

  TELL_NUM ("043","6-34","select emp.name, location from emp, dept");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( EMP.NAME & LOCATION,
    FROM => EMP & DEPT ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         LOCATION");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_LOCATION , LOCATION_LAST );
      T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 044 from page 6-35
--
-- example on page 6-35 is not legal ANSI SQL -- table.* notation is not
--  provided in ANSI SQL
-- select emp.Name, dept.*
-- from   emp, dept
-- where  Dept_No = dept.Number /

  TELL_NUM ("044","6-35","select emp.name, dept.* from emp, " &
       "dept where dept = dept.number");
  PUT_LINE ("This example is not legal ANSI SQL -- table.* notation is not");
  PUT_LINE ("provided in ANSI SQL");
  PUT_LINE ("This example is not executed here");

--
-- 045 from page 6-36
--

  TELL_NUM_2 ("045","6-36",
       "select name, salary * 12, min_amount, max_amount from emp, taxes",
       "       where salary * 12 between min_amount and max_amount");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY ( SALARY * 12.0 ) &
              MIN_AMOUNT & MAX_AMOUNT, -- see previous comments on type
    FROM  => EMP & TAXES,              --  conversions
    WHERE => BETWEEN
             ( CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY ( SALARY * 12.0 ),
               MIN_AMOUNT and MAX_AMOUNT ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         ANNUAL_PAY  MIN_AMOUNT  MAX_AMOUNT");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_ANNUAL_PAY );
      F_FLOAT := FLOAT ( V_ANNUAL_PAY);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (18);
      PUT (F_STRING);
    INTO ( V_MIN_AMOUNT );
      F_FLOAT := FLOAT ( V_MIN_AMOUNT );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (30);
      PUT (F_STRING);
    INTO ( V_MAX_AMOUNT );
      F_FLOAT := FLOAT ( V_MAX_AMOUNT);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (42);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 046 from page 6-37
-- here is another example of a computation that becomes laborious with
--  strong typing.  But, any Ada program applying typing, whether going to a
--  database or not, will have to perform the same type conversions to keep
--  the types meaningful
--

  TELL_NUM_5 ("046","6-37",
       "select emp.name, location, salary * 12 + commission, ",
       "       base_tax + (((salary * 12) +Wcommission) - min_amount) * " &
            "marginal_rate",
       "           from emp, taxes, dept ",
       "           where (salary * 12) + commission between min_amount " &
            "and max_amount",
       "           and dept = dept.number");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( EMP.NAME & LOCATION &
              CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY -- see previous comments
              ( SALARY * 12.0 + COMMISSION ) &      --  about type conversions
              ( BASE_TAX +
                 CONVERT_TO.EXAMPLE_TYPES.TAX_AMOUNT
                 ( CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
                   ( CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY
                     ( SALARY * 12.0 + COMMISSION ) - MIN_AMOUNT ) *
                   CONVERT_TO.EXAMPLE_TYPES.TAX_COMPUTATION_PRECISION
                   ( MARGINAL_RATE ) ) ),
    FROM  => EMP & TAXES & DEPT,                     -- qualification of CODE on
    WHERE => BETWEEN                                 --  last line is not
             ( CONVERT_TO.EXAMPLE_TYPES.ANNUAL_PAY --  required since we
               ( SALARY * 12.0 + COMMISSION ),     --  changed the column
               MIN_AMOUNT and MAX_AMOUNT )         --  name, but is retained
    AND      EQ ( DEPT , DEPT.CODE ) ) );          --  to track example

  OPEN ( CURSOR );

begin
  PUT_LINE (
   "EMP_NAME         LOCATION          (SALARY*12)+COMMISSION  TAX_AMOUNT");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_LOCATION , LOCATION_LAST );
      T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_ANNUAL_PAY );
      F_FLOAT := FLOAT ( V_ANNUAL_PAY);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (42);
      PUT (F_STRING);
    INTO ( V_BASE_TAX );
      F_FLOAT := FLOAT ( V_BASE_TAX);
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (59);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 047 from page 6-38
--

  TELL_NUM_2 ("047","6-38",
     "select emp.name, emp.salary, mgr.name, mgr.salary from emp, mgr.emp",
     "where emp.salary >= mgr.salary and emp.manager = mgr.number");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( EMP.NAME & EMP.SALARY & MGR.NAME & MGR.SALARY,
    FROM  => EMP & MGR.EMP,
    WHERE => EMP.SALARY >= MGR.SALARY
    AND      EQ ( EMP.MANAGER , MGR.NUMBER ) ) );

  OPEN ( CURSOR );
```

```
begin
  PUT_LINE ("EMP_NAME             SALARY   MGR_NAME         MGR_SALARY");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (18);
      PUT (F_STRING);
    INTO ( V_MGR_NAME , STR_LAST_2 );
      T_LEN := INTEGER (STR_LAST_2 - V_MGR_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_MGR_NAME
            (V_MGR_NAME'FIRST + I - 1));
      end loop;
      SET_COL (30);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_MGR_SALARY );
      F_FLOAT := FLOAT ( V_MGR_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (47);
      PUT (F_STRING);
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 048 from page 6-38
--

  TELL_NUM_4 ("048","6-38",
     "select emp.name, dept.location, mgr.name, mgr_dept.location",
     "from emp, dept, mgr.emp, mgr_dept.dept where emp.manager = mgr.number",
     "and emp.dept = dept.code and mgr.dept = mgr_dept.code",
     "and ^= dept.location = mgr_dept.location");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( EMP.NAME & DEPT.LOCATION & MGR.NAME & MGR_DEPT.LOCATION,
    FROM  => EMP & DEPT & MGR.EMP & MGR_DEPT.DEPT,
    WHERE => EQ ( EMP.MANAGER , MGR.NUMBER )
    AND      EQ ( EMP.DEPT , DEPT.CODE )
    AND      EQ ( MGR.DEPT , MGR_DEPT.CODE )
    AND      NE ( DEPT.LOCATION , MGR_DEPT.LOCATION ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME          LOCATION         MGR_NAME         MGR_LOCATION");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_LOCATION , LOCATION_LAST );
      T_LEN := INTEGER (LOCATION_LAST - V_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_LOCATION (V_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_MGR_NAME , STR_LAST_2 );
      T_LEN := INTEGER (STR_LAST_2 - V_MGR_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_MGR_NAME
            (V_MGR_NAME'FIRST + I - 1));
      end loop;
      SET_COL (35);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_MGR_LOCATION , LOCATION_LAST_2 );
      T_LEN := INTEGER (LOCATION_LAST_2 - V_MGR_LOCATION'FIRST + 1);
      for I in 1..T_LEN loop
        DLI := DEPT_LOC_INDEX (I);
        T_STRING (I) := CHARACTER (V_MGR_LOCATION
                 (V_MGR_LOCATION'FIRST + DLI - 1));
      end loop;
      SET_COL (52);
      PUT (T_STRING (1..T_LEN));
  end loop;
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
```

```
-- 049 from page 6-40
--

  TELL_NUM_5 ("049","6-40",
     "select name, job, salary + commission from x.emp where job is in",
     "       select job from emp group by job having count (*) >= 4;",
     "and salary + commission > ",
     "       select sum (salary + commission) * 0.25 from emp " &
           "where emp.job = x.job",
     "              and emp.number ^= x.number group by job");

  DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC  ( NAME & JOB & ( SALARY + COMMISSION ),
    FROM  => X.EMP,
    WHERE => IS_IN ( JOB,
      SELEC     ( JOB,
      FROM     => EMP,
      GROUP_BY => JOB,
      HAVING   => COUNT('*') >= 4 ) )
    AND      SALARY + COMMISSION >
      SELEC     ( SUM ( SALARY + COMMISSION ) * 0.25,
      FROM     => EMP,
      WHERE    => EQ ( EMP.JOB , X.JOB )
      AND         NE ( EMP.NUMBER , X.NUMBER ),
      GROUP_BY => JOB ) ) );

  OPEN ( CURSOR );

begin
  PUT_LINE ("EMP_NAME         JOB              SALARY+COMMISSION");
  loop
    FETCH ( CURSOR );
    INTO ( V_EMP_NAME , STR_LAST );
      T_LEN := INTEGER (STR_LAST - V_EMP_NAME'FIRST + 1);
      for I in 1..T_LEN loop
        T_STRING (I) := CHARACTER (V_EMP_NAME (V_EMP_NAME'FIRST + I - 1));
      end loop;
      SET_COL (1);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_JOB , JOB_LAST );
      T_LEN := INTEGER (JOB_LAST - V_JOB'FIRST + 1);
      T_STRING (1..T_LEN) := STRING (V_JOB (V_JOB'FIRST .. JOB_LAST));
      SET_COL (18);
      PUT (T_STRING (1..T_LEN));
    INTO ( V_SALARY );
      F_FLOAT := FLOAT ( V_SALARY );
      FLOAT_TO_STRING (F_FLOAT, F_STRING);
      SET_COL (35);
      PUT (F_STRING);
  end loop;
```

```
exception
  when NOT_FOUND_ERROR => null;
end;

  CLOSE ( CURSOR );

--
-- 050 from page 6-41
--

  TELL_NUM ("050","6-41 ", "insert into dept (number, name, location) " &
       ": < COLL, 'Collection', 'Atlanta' >");

  INSERT_INTO ( DEPT ( CODE & NAME & LOCATION ),
  VALUES <= COLL and EXAMPLE_TYPES.ADA_SQL.DEPT_NAME'("Collection     ")
             and EXAMPLE_TYPES.ADA_SQL.DEPT_LOC'("Atlanta") );

-- Note that literals may require type qualification if their type could be
--  ambiguous.  In real programs, program variables would typically be used
--  rather than literals, so type qualification would, in general, not be
--  required.

--
-- 051 from page 6-42
--
-- example on page 6-42 is not legal ANSI SQL -- cannot insert more than one
--  row at a time
-- insert into emp:
--   < 3000, 'Owens',    80, 'clerk',  3100,  950.00, 0.00 >,
--   < 3100, 'Clark',    80, 'c.p.a.', 2400, 1800.00, 0.00 >,
--   < 3200, 'Williams', 80, 'clerk',  3100, 2500.00, 0.00 > /

  TELL_NUM_4 ("051 ","6-42","insert into emp:",
     "     < 3000, 'Owens',    COLL, 'clerk',  3100,  950.00, 0.00 >,",
     "     < 3100, 'Clark',    COLL, 'c.p.a.', 2400, 1800.00, 0.00 >,",
     "     < 3200, 'Williams', COLL, 'clerk',  3100, 2500.00, 0.00 >");
  PUT_LINE ("This example is not legal ANSI SQL -- " &
       "cannot insert more than one row at a time");
  PUT_LINE ("This example is broken into three examples, 52, 53 & 54");

--
-- 052 from page 6-42
--

  TELL_NUM_2 ("052","6-42","insert into emp:",
     "     < 3000, 'Owens',    COLL, 'clerk',  3100,  950.00, 0.00 >");

  INSERT_INTO ( EMP ,
  VALUES <= EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(3000) and
            EXAMPLE_TYPES.ADA_SQL.EMP_NAME'("Owens     ") and
            COLL and
            EXAMPLE_TYPES.ADA_SQL.EMP_JOB'("clerk      ") and
            EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(3100) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(950.00) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(0.00) );

--
-- 053 from page 6-42
--

  TELL_NUM_2 ("053","6-42","insert into emp:",
     "     < 3100, 'Clark',    COLL, 'c.p.a.', 2400, 1800.00, 0.00 >");

  INSERT_INTO ( EMP,
  VALUES <= EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(3100) and
            EXAMPLE_TYPES.ADA_SQL.EMP_NAME'("Clark      ") and
            COLL and
            EXAMPLE_TYPES.ADA_SQL.EMP_JOB'("c.p.a.     ") and
            EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(2400) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(1800.00) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(0.00) );

--
-- 054 from page 6-42
--

  TELL_NUM_2 ("054","6-42","insert into emp:",
     "     < 3200, 'Williams', COLL, 'clerk',  3100, 2500.00, 0.00 >");

  INSERT_INTO ( EMP,
  VALUES <= EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(3200) and
            EXAMPLE_TYPES.ADA_SQL.EMP_NAME'("Williams  ") and
            COLL and
            EXAMPLE_TYPES.ADA_SQL.EMP_JOB'("clerk.     ") and
            EXAMPLE_TYPES.ADA_SQL.EMP_NUMBER'(3100) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(2500.00) and
            EXAMPLE_TYPES.ADA_SQL.MONTHLY_PAY'(0.00) );

--
-- 055 from page 6-42
--

  TELL_NUM_2 ("055","6-42","insert into candidates:",
     "  select number, name, dept, salary from emp where " &
     "commission > 0.5 * salary");

  INSERT_INTO ( CAND,
    SELEC  ( NUMBER & NAME & DEPT & SALARY,
    FROM  => EMP,
    WHERE => COMMISSION > 0.5 * SALARY ) );
```

```
--
-- 056 from page 6-43
--

  TELL_NUM ("056","6-43",
       "update emp set salary = 1000 where name = 'Owens-----'");

  UPDATE ( EMP,                            -- cannot use "Owens*" for Ada
  SET   => SALARY <= 1000.0,               --  reasons -- we made EMP_NAME
  WHERE => LIKE ( NAME , "Owens_____" ) ); --  constrained to be illustrative

--
-- 057 from page 6-43
--

  TELL_NUM ("057","6-43","update emp set commission = salary * 0.35, " &
       "DEPT = FIN where job = 'salesman%'");

  UPDATE ( EMP,
  SET   => COMMISSION <= SALARY * 0.35
       and DEPT <= FIN,
  WHERE => LIKE ( JOB , "salesman*" ) );

--
-- 058 from page 6-43
--
-- example on page 6-43 is not legal ANSI SQL -- cannot use subquery to
--  produce values for update
-- update emp
-- set Commission =
--   select Base_Tax + ((Salary * 12) - Min_Amount) * Marginal_Rate
--   from   emp x, taxes
--   where  Salary*12 between Min_Amount and Max_Amount
--   and    emp.Number = x.Number;
-- where Dept_No = 70 /

  TELL_NUM_4 ("058","6-43","update emp set commission =",
     "select base_tax + ((salary * 12) - min_amount) * marginal_rate",
     "from x.emp, taxes where salary * 12 between min_amount and max_amount",
     "and emp.number = x.number; where dept = FIN");
  PUT_LINE ("This example is not legal ANSI SQL -- cannot use " &
       "subquery to produce");
  PUT_LINE ("values for update");
  PUT_LINE ("This example is not executed here");

--
-- 059 from page 6-44
--

  TELL_NUM ("059","6-44","delete emp where name = 'Owens_____'");
  DELETE_FROM ( EMP,
  WHERE => LIKE ( NAME , "Owens_____" ) ); -- see above comment on "Owens*"

--
-- 060 from page 6-44
--

  TELL_NUM ("060","6-44","delete emp where dept = select code from " &
      "dept where location = 'Atlanta%'");

  DELETE_FROM ( EMP,
  WHERE => EQ ( DEPT,
    SELEC  ( CODE,
    FROM  => DEPT,
    WHERE => LIKE ( LOCATION , "Atlanta*" ) ) ) );

end EXAMPLE_2;

end EX_2;
```

15. Procedure EXAMPLE

```
with TEXT_IO, SYSTEM, EXAMPLE_TYPES, EXAMPLE_VARIABLES, EX_1, EX_2;
use  TEXT_IO, SYSTEM, EXAMPLE_TYPES, EXAMPLE_VARIABLES, EX_1, EX_2;

procedure EXAMPLE is
use  EXAMPLE_TYPES.ADA_SQL;

    subtype ADDRESS is SYSTEM.ADDRESS;
    procedure CSYSTEM (STR : ADDRESS);
    pragma Interface (C, CSYSTEM);
    TMP     : STRING(1..62) :=
        "sh /div/brykczynski/fill.exdb >/div/brykczynski/fill.out 2>&1 " &
        ascii.nul;

begin
  V_NUMBER              := 1 ;
  V_EMP_NAME            := "          " ;
  V_DEPT                := ADMIN ;
  V_JOB                 := "          " ;
  V_MANAGER             := 1 ;
  V_SALARY              := 0.0 ;
  V_MAX_SALARY          := 0.0 ;
  V_COMMISSION          := 0.0 ;
  V_MINIMUM_COMMISSION  := 0.0 ;
  V_DEPT_NAME           := "               " ;
  V_LOCATION            := "               " ;
```

```
V_MIN_AMOUNT          := 0.0 ;
V_MAX_AMOUNT          := 0.0 ;
V_BASE_TAX            := 0.0 ;
V_EXTRA_TAX           := 0.0 ;
V_ANNUAL_PAY          := 0.0 ;
V_MARGINAL_RATE       := 0.0 ;
V_TOTAL_PAY           := 0.0 ;
V_MGR_NAME            := "          " ;
V_MGR_SALARY          := 0.0 ;
V_MGR_LOCATION        := "               " ;
COUNT_RESULT          := 0 ;
STR_LAST              := 1 ;
STR_LAST_2            := 1 ;
JOB_LAST              := 1 ;
LOCATION_LAST         := 1 ;
LOCATION_LAST_2       := 1 ;


PUT_LINE ("This Ada/SQL application program executes the " &
        "database functions shown in the");
PUT_LINE ("examples in the Unify Reference Manual, Section 6 " &
        "'SQL - Query/DML Language'.");
PUT_LINE ("The Unify database used must be filled with the " &
        "correct data when this program");
PUT_LINE ("begins to execute.  To prime the database we will now " &
        "run the command file ");
PUT_LINE ("/div/brykczynski/fill.exdb.  Output will go to " &
        "/div/brykczynski/fill.out.");
PUT_LINE ("You don't need to check this file unless you want to.");
PUT_LINE (" ");
CSYSTEM (TMP'ADDRESS);
PUT_LINE ("This example uses four database tables " &
        "EMP, DEPT, TAXES and CAND from the Unify");
PUT_LINE ("database.  Below is a list of the names used " &
        "to reference the fields in the");
PUT_LINE ("Ada/SQL program and the Unify manual.");
PUT_LINE (" ");
PUT_LINE (" ");
PUT_LINE ("table: EMP   Ada/SQL       Unify");
PUT_LINE ("             NUMBER        NUMBER");
PUT_LINE ("             NAME          NAME");
PUT_LINE ("             DEPT          DEPT_NO");
PUT_LINE ("             JOB           JOB");
PUT_LINE ("             MANAGER       MANAGER");
PUT_LINE ("             SALARY        SALARY");
PUT_LINE ("             COMMISSION    COMMISSION");
PUT_LINE (" ");
PUT_LINE ("table: DEPT  Ada/SQL       Unify");
PUT_LINE ("             CODE          NUMBER");
PUT_LINE ("             NAME          NAME");
```

```
  PUT_LINE ("                LOCATION      LOCATION");
  PUT_LINE (" ");
  PUT_LINE ("table: TAXES Ada/SQL       Unify");
  PUT_LINE ("             MIN_AMOUNT    MIN_AMOUNT");
  PUT_LINE ("             MAX_AMOUNT    MAX_AMOUNT");
  PUT_LINE ("             BASE_TAX      BASE_TAX");
  PUT_LINE ("             MARGINAL_RATE MARGINAL_RATE");
  PUT_LINE (" ");
  PUT_LINE ("table: CAND  Ada/SQL       Unify");
  PUT_LINE ("             NUMBER        NUMBER");
  PUT_LINE ("             NAME          NAME");
  PUT_LINE ("             DEPT          DEPT_NO");
  PUT_LINE ("             SALARY        SALARY");
  PUT_LINE (" ");
  PUT_LINE (" ");
  PUT_LINE ("All of the data types in these tables are the " &
        "same except for DEPT of EMP,");
  PUT_LINE ("CODE of DEPT and DEPT of CAND, which all refer to " &
        "the department field.  The");
  PUT_LINE ("Unify manual treats these columns as numeric with " &
        "values 10, 20, 30, 40, 50,");
  PUT_LINE ("60, 70 and 80.  The database we're using defines a " &
        "numeric field with the");
  PUT_LINE ("values of 1, 2, 3, 4, 5, 6, 7 and 8.  The Ada/SQL " &
        "example program treats these");
  PUT_LINE ("fields as an enumeration type where 1 = ADMIN, " &
        "2 = ESALES, 3 = CSALES, 4 =");
  PUT_LINE ("WSALES, 5 = MKTING, 6 = RSRCH, 7 = FIN, and 8 = COLL");
  PUT_LINE (" ");

  EXAMPLE_1;
  EXAMPLE_2;
end EXAMPLE;
```

16. Sample Data

This portion of the software contains two sets of SQL, each of which will be invoked by the Ada/SQL system. The DELETE's will be invoked to clear out any data which may have been left in the UNIFY database due to a prior execution. the FILL's will bulk load the database with the data which will be used in the demonstration of the Ada/SQL system.

```
delete CAND where NUMBER < 9999 /
delete customer where Customer_Number < 9 /
delete DEPT where CODE < 9999 /
delete EMP where NUMBER < 9999 /
delete item where Serial_Number < 9999 /
delete manf where Manufacturer_ID < 999 /
delete model where Model_Number < 99999 /
delete TAXES where MARGINAL_RATE < 9.999 /
```

```
file FILL.ALL

SQL del.taxes
SQL del.emp
SQL del.dept
SQL del.cand
SQL fill.dept
SQL fill.emp
SQL fill.taxes
SQL del.taxes
SQL del.emp
SQL del.dept
SQL del.cand
SQL fill.dept
SQL fill.emp
SQL fill.taxes

file FILL.CAN

insert into CAND:
< 1900, 'Brown',     60, 2000.00 > /
insert into CAND:
< 2800, 'Fiorella',  70, 800.00 > /
insert into CAND:
< 1800, 'Amato',     40, 2000.00 > /
insert into CAND:
< 2700, 'Colucci',   40, 2500.00 > /
insert into CAND:
< 1700, 'Moehr',     70, 950.00  > /
insert into CAND:
< 2600, 'Bleriot',   10, 1100.00  > /
insert into CAND:
< 1600, 'Dupre',     50, 800.00   > /
insert into CAND:
< 2500, 'Kawasaki',  30, 1800.00 > /
insert into CAND:
< 1500, 'Otsaki',    60, 1800.00  > /
insert into CAND:
< 2400, 'Lee',       10, 7500.00  > /
insert into CAND:
< 1400, 'Scharf',    10, 800.00   > /
insert into CAND:
< 2300, 'Klein',     20, 1500.00  > /
insert into CAND:
< 1300, 'Schmidt',   60, 2500.00  > /
insert into CAND:
< 2200, 'Dugan',     40, 1650.00  > /
insert into CAND:
< 1200, 'O"Neil',   20, 1500.00  > /
```

```
insert into CAND:
< 2100, 'Reilly',    30, 2500.00 > /
insert into CAND:
< 1100, 'Whittaker', 20, 2500.00  > /
insert into CAND:
< 2000, 'Jones',     10, 900.00   > /
insert into CAND:
< 1000, 'Smith',     50, 1500.00 > /

file FILL.CUS

insert into customer:
<  3, 'Reliable Construction Co.', '2113 Folsom Blvd.', 'Sacramento', 'CA',
insert into customer:
<  2, 'Creative manufacturing', '9124 Industrial Blvd.', 'Redding', 'CA',
insert into customer:
<  1, 'Smith & Sons Hardware', '1234 State Street', 'Wheatville', 'CA',

file FILL.DEP

insert into DEPT:
< 7, 'Finance',         'Dallas' > /
insert into DEPT:
< 6, 'Research',        'Dallas' > /
insert into DEPT:
< 5, 'Marketing',       'San Francisco' > /
insert into DEPT:
< 4, 'Western Sales',   'Los Angeles' > /
insert into DEPT:
< 3, 'Central Sales',   'Chicago' > /
insert into DEPT:
< 2, 'Eastern Sales',   'New York' > /
insert into DEPT:
< 1, 'Administration', 'Dallas' > /

file FILL.EMP

insert into EMP:
< 1900, 'Brown',     6, 'engineer',   1300,  2000.00,    0.00 > /
insert into EMP:
< 2800, 'Fiorella',  7, 'clerk',      1700,   800.00,    0.00 > /
insert into EMP:
< 1800, 'Amato',     4, 'salesman',   2200,  2000.00,  750.00 > /
insert into EMP:
< 2700, 'Colucci',   4, 'salesman',   2200,  2500.00, 3000.00 > /
insert into EMP:
< 1700, 'Moehr',     7, 'clerk',      2400,   950.00,    0.00 > /
insert into EMP:
< 2600, 'Bleriot',   1, 'programmer', 1300,  1100.00,    0.00 > /
insert into EMP:
```

```
< 1600, 'Dupre',     5, 'clerk',       1000,   800.00,    0.00 > /
insert into EMP:
< 2500, 'Kawasaki',  3, 'salesman',    2100,  1800.00, 1000.00 > /
insert into EMP:
< 1500, 'Otsaka',    6, 'engineer',    1300,  1800.00,    0.00 > /
insert into EMP:
< 2400, 'Lee',       1, 'president',   2400,  7500.00,    0.00 > /
insert into EMP:
< 1400, 'Scharf',    1, 'clerk',       2000,   800.00,    0.00 > /
insert into EMP:
< 2300, 'Klein',     2, 'salesman',    1100,  1500.00,    0.00 > /
insert into EMP:
< 1300, 'Schmidt',   6, 'programmer',  2400,  2500.00,    0.00 > /
insert into EMP:
< 2200, 'Dugan',     4, 'salesman',    2400,  1650.00,  900.00 > /
insert into EMP:
< 1200, 'O"Neil',    2, 'salesman',    1100,  1500.00,  150.00 > /
insert into EMP:
< 2100, 'Reilly',    3, 'salesman',    2400,  2500.00, 1500.00 > /
insert into EMP:
< 1100, 'Whittaker', 2, 'salesman',    2400,  2500.00,  500.00 > /
insert into EMP:
< 2000, 'Jones',     1, 'clerk',       1300,   900.00,    0.00 > /
insert into EMP:
< 1000, 'Smith',     5, 'salesman',    2100,  1500.00, 1000.00  > /

file FILL.ITE

insert into item:
< 1234,1002,100, 02/23/84,15.75,2,13.49 >  /
insert into item:
< 1013,1012,104, 02/08/84, 9.97,2, 9.23 >  /
insert into item:
< 1012,1011,104, 02/08/84, 5.77,1, 5.75 >  /
insert into item:
< 1011,91117,103,02/15/84, 3.35,1, 2.20 >  /
insert into item:
< 1010,91117,103,01/15/84, 3.03,1, 3.00 > /
insert into item:
< 1009,61117,103,02/15/84, 2.82,2,2.00 >  /
insert into item:
< 1008,61117,103,01/15/84, 2.47,2,2.20 >  /
insert into item:
< 1007, 1002,102,01/19/84, 9.19,1,5.25 >  /
insert into item:
< 1006,55271,101,02/15/84, 7.23,1,6.00 >  /
insert into item:
< 1005,55071,101,02/25/84,11.29,3,9.25 >  /
insert into item:
< 1004,55071,101,02/23/84,10.76,1,8.90 >  /
```

```
insert into item:
< 1003, 1001,100,02/15/84,10.24,1,7.25 >  /
insert into item:
< 1002, 1001,100,02/15/84,10.24,3,7.35 >  /
insert into item:
< 1001, 1001,100,02/15/84,10.24,1,7.25 > /

file FILL.MAN

insert into manf :
< 105,'BHP Ltd.','17385 Weatherby Rd.','New York','NY',10022 > /
insert into manf :
< 104,'The Tool Depot','7562 Orange Dr.','Tucson','AZ',85745 > /
insert into manf :
< 103,'Grover Parts and Supplies','9462 Jackson Road','Rancho Cordova',
insert into manf :
< 102,'A & H Industries, Inc.','2434 Evergreen Ave.','Eagan','MN',55422 > /
insert into manf :
< 101,'Precision Tool Co.','2600 West 16th Street','San Francisco',
insert into manf :
< 100,'RH Smith Manufacturing','523 Galveston Ave.','Centerville','CA',95923 > /

file FILL.MOD

insert into model:
<  1012   ,  104,  '1/2" socket wrench'  > /
insert into model:
<  1011   ,  104,  'combination pliers'  > /
insert into model:
<  91117  ,  103,  '6" slotted screwdriver'  > /
insert into model:
<  81117  ,  103,  '6" Phillips screwdriver'  > /
insert into model:
<  71117  ,  103,  '3" slotted screwdriver'  > /
insert into model:
<  61117  ,  103,  '3" Phillips screwdriver'  > /
insert into model:
<  1002   ,  102,  'leather mallet'  > /
insert into model:
<  1001   ,  102,  'vise grips'  > /
insert into model:
<  55371  ,  101,  'needle nose pliers'  > /
insert into model:
<  55271  ,  101,  'combination pliers'  > /
insert into model:
<  55171  ,  101,  '1/2" box end wrench'  > /
insert into model:
<  55071  ,  101,  '1/2" socket wrench'  > /
insert into model:
<  1003   ,  100,  '1/2" open end wrench'  > /
```

```
insert into model:
<  1002   ,  100,  '3/4" socket wrench'  > /
insert into model:
<  1001   ,  100,  '1/2" socket wrench'  > /


file FILL.ORD

insert into orders:
< 0, **/**/**, 0 > /
insert into orders:
< 3, 06/28/82, 2 > /
insert into orders:
< 2, 04/02/82, 3 > /
insert into orders:
< 1, 04/02/82, 1 > /

file FILL.TAX

insert into TAXES:
< 85600.00, 99999.00, 30249.00, 0.50000 > /
insert into TAXES:
< 60000.00, 85600.00, 17705.00, 0.49000 > /
insert into TAXES:
< 45800.00, 60000.00, 11457.00, 0.44000 > /
insert into TAXES:
< 35200.00, 45800.00,  7323.00, 0.39000 > /
insert into TAXES:
< 29900.00, 35200.00,  5574.00, 0.33000 > /
insert into TAXES:
< 24600.00, 29900.00,  4037.00, 0.29000 > /
insert into TAXES:
< 20200.00, 24600.00,  2937.00, 0.25000 > /
insert into TAXES:
< 16000.00, 20200.00,  2013.00, 0.22000 > /
insert into TAXES:
< 11900.00, 16000.00,  1234.00, 0.19000 > /
insert into TAXES:
<  7600.00, 11600.00,   546.00, 0.16000 > /
insert into TAXES:
<  5500.00,  7600.00,   252.00, 0.14000 > /
insert into TAXES:
<  3400.00,  5500.00,     0.00, 0.12000 > /
insert into TAXES:
<     0.00,  3400.00,     0.00, 0.00000 > /
```

Distribution List for IDA Memorandum Report M-361

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
Ms. Sally Barnes HQ Defense Logistics Agency (DLA) ATTN DLA-ZWS Cameron Station Alexandria, VA 22304-6100	10 copies
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
IIT Research Institute 4550 Forbes Blvd., Suite 300 Lanham, MD 20706	1 copy
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22003	1 copy
Ms. Patty Hicks DSAC-SR 3990 Broad St. Columbus, OH 43216-5002	1 copy
Ms. Kerry Hilliard 7321 Franklin Road Annandale, VA 22003	1 copy
Ms. Elinor Koffee DSAC-SR 3990 Broad St. Columbus, OH 43216-5002	1 copy
CSED Review Panel	
Dr. Dan Alpert, Director Center for Advanced Study University of Illinois 912 W. Illinois Street Urbana, Illinois 61801	1 copy

NAME AND ADDRESS	NUMBER OF COPIES
Dr. Barry W. Boehm TRW Defense Systems Group MS 2-2304 One Space Park Redondo Beach, CA 90278	1 copy
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1 copy
Dr. Larry E. Druffel Software Engineering Institute Shadyside Place 480 South Aiken Av. Pittsburgh, PA 15231	1 copy
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1 copy
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1 copy
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1 copy
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1 copy
IDA	
General W.Y. Smith, HQ	1 copy
Mr. Seymour Deitchman, HQ	1 copy
Mr. Philip Major, HQ	1 copy
Dr. Jack Kramer, CSED	1 copy
Dr. Robert I. Winner, CSED	1 copy
Dr. John Salasin, CSED	1 copy
Mr. Bill Brykczynski, CSED	10 copies
Ms. Audrey A. Hook, CSED	2 copies
Ms. Katydean Price, CSED	2 copies
IDA Control & Distribution Vault	3 copies